Universidad Carlos III de Madrid

Escuela politécnica superior

Departamento de Telemática

Ingeniería técnica de Telecomunicación especialidad Sonido e Imagen

Proyecto de Fin de Carrera

Desarrollo de un sistema de televigilancia a través de un dispositivo móvil

Autor: Javier Rodríguez Camarma

Tutor: Mario Muñoz Organero

Noviembre 2009

Título: Desarrollo de un sistema de televigilancia a través de un dispositivo móvil.

Autor: Javier Rodríguez Camarma

Tutor: Mario Muñoz Organero

EL TRIBUNAL

Presidente:

Secretario:

Vocal:

Realizado el acto de defensa del Proyecto Fin de Carrera el día __ de _____ de 2009 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Fdo: Presidente Fdo: Secretario Fdo: Vocal

Resumen

El Proyecto Fin de Carrera **"Desarrollo de un sistema de televigilancia a través de un dispositivo móvil"** analiza e implementa una aplicación de televigilancia utilizando un dispositivo móvil como elemento de captura.

Las funciones principales que implementa este sistema son: detección de movimiento en las imágenes tomadas por la cámara, configuración de seguridad y números de envío, comunicación mediante mensajería (SMS y MMS) y bluetooth y también con un servidor Web.

El principal objetivo de este proyecto es desarrollar una aplicación capaz de dictaminar cunado se produce un movimiento en las imágenes tomadas con la cámara y automáticamente enviar un mensaje multimedia con dicha imagen a otro terminal.

Agradecimientos

Gracias a vuestro a apoyo he conseguido concluir esta etapa y en estas palabras se encuentran mis agradecimientos a los verdaderos protagonistas de este cuento.

A mis padres, **José María** y **Concepción**, y mi hermano **José María**, por permitirme andar el camino.
Al resto de familiares por contribuir de una manera u otra en ese camino, en especial a **Irene**.
A **Móstoles** por verme crecer y a **Cabanillas** por ayudar en el proceso.

A **Adrián** por apoyarme siempre aunque no llevásemos buena mano; a **David** por conducirme por el buen camino; a **Jose**, por mostrarme que con la mayor de las pérdidas, sin duda, salí ganando; sin olvidarme de **Jess**, **Paula**, **Nuria** y **Leti**.

A Mario, nuestro tutor, por darnos esta oportunidad.

A mi compañero Juan (**Iru**), porque trabajar con él es un lujo y un honor. A **Ana Belén**, por comprender el silencio; a **Clara**, por estar junto a mí, incluso en lugares que nunca hubiera imaginado; a **Cris**, porque la sinceridad es un bien escaso; a **Joven** (Carlos) por saber cuándo empieza una conversación (o monólogo), y no querer que termine; a **Lore**, por el tiempo que el tiempo olvido; a **Myri**, por estar aunque no sea físicamente; a **Rubén**, por conocerte despistado y no acordarte nunca de cambiar; a **Tony** porque cada palabra puede ser divertida.

Al resto de Sonimágicos: **Alvarito** y **Pablo** (mis hijos adoptivos), **Ana Román**, **Alex**, **Carlicos**, **Crespo**, **Cris del Pozo**, **Dani10**, **Félix**, **Herrera**, **JC**, Jesús (**Pequeñín**), **Lucía**, **Marian**, **Moni**, Patri (**Pato**), **Pisha**, **Sarita**, **Soriano**, **Vaca**; porque me gustaría dedicaros más espacio, pero no dispongo de él. A los que falten por comprender a mi escasa memoria

A **Antonio** (Academia Pepe), porque el camino es duro y complicado y él lo ha hecho más sencillo y sobre todo, muy divertido.

A La Velilla, Valdeolmos, Valdelcubo, Cañavate, El Hito, Otero de Herreros, Cancún, Salamanca, Denia, Goteborg, Ibiza porque siempre "**Volverá**" a "**Volare**".

A todos aquellos que han intentado que este proyecto no llegara a buen fin, porque sin duda, han contribuido a que me esfuerce más, y como no, al **27**.

GRACIAS

Índice de contenidos

1. Introducción y objetivos

1.1. Marco

Este proyecto se encuentra dentro del desarrollo de un proyecto global "**Desarrollo e integración de un sistema de televigilancia a través de un dispositivo móvil en un servidor Web**".

El proyecto global busca la creación de un sistema de televigilancia en la que el elemento de captación es un teléfono móvil (concretamente su cámara) y en el que el sistema de almacenamiento es un servidor Web.

Esta memoria versa sobre el sistema basado en el teléfono móvil, "**Desarrollo de un sistema de tele-vigilancia a través de un dispositivo móvil**." En ella podrá encontrar:

- Un estudio de los dispositivos móviles actuales y su historia, la tecnología utilizada para el desarrollo de aplicaciones móviles y la evolución de los sistemas de televigilancia [Estado del arte, apartado 2].

- El análisis de las características que debe cumplir la aplicación en su ejecución [Análisis de la aplicación, apartado 3].

- La descripción técnica de cómo se ha implementado la aplicación [Descripción de la aplicación, apartado 4].

- Un análisis de las posibles mejoras a realizar en la aplicación y los trabajos a desarrollar en base a ella [Posibles mejoras y trabajos futuros, apartado 5].

- Las conclusiones obtenidas al finalizar el desarrollo de la aplicación [Conclusiones, apartado 6].

- Una serie de anexos para completar al proyecto:
  - Una visión de la carga de trabajo asociada al proyecto, tanto en su desarrollo como en su puesta en marcha [Planificación, anexo I].
  - Diagramas de flujo que sirve de apoyo al desarrollo de la aplicación [Diagramas de flujo, anexo II].
  - Una explicación de los pasos a seguir para que un usuario instale y utilice esta aplicación [Guía de uso, anexo III].

El estudio, desarrollo y explicación del servidor Web se encuentra en el texto **"Integración de un sistema de televigilancia a través de un dispositivo móvil en un servidor Web"**, llevado a cabo por Juan Galán García.

1.2. Introducción

Hoy en día un teléfono móvil no es solo un dispositivo con el cual realizar llamadas y enviar mensajes. Se trata de un auténtico banco multimedia cada vez más cercano a un ordenador. A la hora de adquirir un nuevo teléfono móvil esperamos tener una agenda lo más completa posible, una cámara con unas características que suplanten a una real, un reproductor multimedia de gran capacidad; y todo ello en un espacio mínimo y con una gran autonomía, y a un precio razonable.

Como usuarios y clientes, buscamos en un teléfono móvil la mayor funcionalidad posible sin demasiado coste, y cuyo funcionamiento sea sencillo e intuitivo. Los fabricantes compiten en busca del móvil más sencillo y con mayores capacidades, evolucionando desde el gran teléfono conocido como "ladrillo" a los actuales teléfonos táctiles en los que es difícil distinguir si se trata de una PDA con cámara y reproductor multimedia, que hace llamadas, o realmente se trata de un teléfono móvil.

La incorporación de máquinas virtuales dentro de los teléfonos móviles ha hecho posible que se pueda desarrollar casi cualquier tipo de aplicación, desde calculadoras y

relojes mundiales hasta sofisticados juegos que no tienen nada que envidiar a los de cualquier otra plataforma. Abriendo el campo de negocio del teléfono móvil mucho más allá de la tarificación por llamada y mensaje y permitiendo a muchos actores entrar dentro del sector de la telefonía móvil.

Desarrolladores de aplicaciones, programadores de juegos, servidores de archivos multimedia se han incorporado como un agente más dentro del sector, y no se contempla el negocio de un dispositivo móvil sin ellos. Si con un teléfono móvil no se puede descargar un juego, tener la aplicación más novedosa o descargar la canción o video de moda, ese teléfono ya es antiguo y "no sirve".

Con este tipo de servicios, el tráfico dedicado desde o hacia el teléfono móvil ha crecido exponencialmente, superando, con creces las ideas iniciales. Por lo que cobran también, vital importancia las redes sobre las cuales están montados los servicios. Y sin duda, sobre todas ellas, la red de Internet.

Internet y el acceso a ella se considera imprescindible en nuestros días, por tanto una de las características deseables en un teléfono móvil, es el acceso a Internet. Este acceso actualmente se proporciona con tarificaciones especiales de las compañías proveedoras de servicio, o incluso mediante la conexión a redes WIFI públicas. Abriendo el concepto de Internet para el móvil, ya que las paginas y servicios habituales que se prestan a un ordenador no son compatibles en todos los aspectos en un terminal móvil, ya que este posee unas limitaciones que un ordenador no tiene (memoria y tamaño de visualización).

Estas limitaciones hacen que a la hora de desarrollar aplicaciones, se ajusten los lenguajes de programación, apareciendo incluso lenguajes específicos como es el caso de J2ME. Se tratan normalmente de lenguajes que mantienen la esencia de sus padres pero que limitan su funcionalidad eliminado todo aquello que un terminal móvil no es capaz de realizar ni procesar, además de incluir elementos específicos propios del teléfono que no aparecen en un ordenador.

Como lenguaje de programación, su paquete básico no es capaz de cubrir todo el rango de funcionalidad que ofrece un terminal, por ello existen paquetes o APIs específicos para controlar el acceso a recursos particulares de un teléfono móvil, que para un dispositivo sencillo no son necesarios y serían demasiado pesados en memoria, pero que para dispositivos con gran capacidad son totalmente necesarios. La elección sobre su uso dependerá de los terminales a los que irán dedicadas las aplicaciones a desarrollar, siendo de vital importancia definir este aspecto, ya que todas las aplicaciones no funcionan en todos los teléfonos, y en la mayoría de los casos, se trata de un problema de compatibilidad de versiones del lenguaje utilizado.

La comunicación entre terminales se realiza en su gran mayoría mediante la realización de llamadas y el envío de mensajes, si bien no es la única forma de poder transmitir cualquier tipo de información o datos entre ellos. Al ya comentado hecho del acceso a Internet, que permite cualquier tipo de transferencia hay que añadir la capacidad de los dispositivos de establecer conexiones mediante infrarrojos y bluetooth. Así, se pueden establecer comunicaciones entre cualquier tipo de dispositivos (ya sean teléfonos móviles o no) que sean capaces de utilizar este tipo de tecnologías.

Especialmente relevante es la conectividad mediante bluetooth. El infrarrojo necesita contacto visual y no es la situación deseable en la mayoría de los casos, aunque si útil en otros. El bluetooth permite la transferencia de datos dentro de un rango de acción sin contacto directo, y quizá lo más importante, sin coste alguno. Teniendo un terminal con conectividad se puede enviar cualquier tipo de datos a cualquier otro que también tenga conectividad.

Este tipo de conexiones son importantes para la interacción de los dispositivos con los ordenadores, las aplicaciones se programan y desarrollan en ordenadores (mediante emuladores) pero la versión final estará en un teléfono móvil y se necesita hacer llegar la aplicación al teléfono móvil. Para ello los métodos más utilizados son los cables que se proporcionan con el terminal y las conexiones bluetooth. Todo terminal tiene asociado un programa de gestión y sincronización con un PC.

Aunque el desarrollo se haga mediante emuladores, siempre es necesario la prueba en los terminales, ya que no existe un emulador genérico para todos los terminales, y cada terminal móvil tiene una respuesta a la aplicación que en ocasiones se aleja sobremanera de la obtenida en el emulador, dándose casos en los que la aplicación no realiza su función en el terminal, aunque en el emulador el funcionamiento sea el correcto.

La seguridad es una de las grandes preocupaciones de la sociedad actual. Por ello, las empresas de seguridad emplean cada vez sistemas más sofisticados con la intención de disuadir y evitar la entrada ilegal a nuestros hogares. Dentro de estos sistemas podemos distinguir las alarmas tradicionales, que cuando detectan algún tipo de cambio en su entorno comienzan con una sirena, o la videovigilancia, en la que una persona o grupo de personas se encargan del visionado continuo de imágenes.

Estos sistemas requieren el montaje de una infraestructura dentro del hogar o del recinto en el que se vaya utilizar, con sensores o cámaras y una unidad central de control, ya sea para el personal de seguridad, o simplemente para realizar una conexión telefónica con la central de seguridad. Existe un intermediario entre el sistema de seguridad y el usuario asegurado.

Este hecho puede resultar cómodo al usuario, ya que no necesita estar totalmente pendiente de su sistema, ni de su mantenimiento, pero a su vez, en caso de fallo genera una gran alarma ya que no conoce directamente la situación, sino que hay una persona que le llama o le avisa de que algo está pasando en la zona donde está instalado el sistema de seguridad, no es él quien conoce directamente este hecho.

Una falsa alarma provoca una gran preocupación y aunque los sistemas actuales son muy fiables, siempre queda la incógnita del posible fallo, quizá por desconfianza ante el sistema, y con ello la incertidumbre de si realmente se ha producido un fallo o la alarma es verdadera.

Con un sistema en tiempo real y sin intermediarios sería el propio usuario el que recibiese el aviso y actuara en consecuencia, si se ha producido un fallo en la detección

tendría una imagen real de lo que está ocurriendo y de lo que ha provocado la alarma, y en un caso real, sería el primero en conocerlo y el que decidiera a qué tipo de estamento de emergencia quiere avisar, sin estar atado a ningún tipo de compañía ni servicio de seguridad, ni a su coste.

1.3. Objetivos

El objetivo de este proyecto es el desarrollo sobre un sistema multimedia universal cotidiano, al alcance de la gran mayoría de la población (teléfono móvil), de un servicio en tiempo real de seguridad, sencillo, fiable, sin ningún tipo de intermediario y a muy bajo coste, que aporte confianza al usuario de que en caso de producirse una situación anómala, será el primero en conocerlo de manera privada en cualquier lugar donde se encuentre.

Este sistema no es infalible, no va a evitar que se produzcan los robos, sería un hecho utópico, pero al tratarse de un sistema silencioso, aumenta la posibilidad de localizar a los intrusos en plena acción (pudiendo evitar así futuros robos). Abaratará exponencialmente el coste del sistema de televigilancia clásico, ya que un teléfono móvil es una herramienta diaria y la tarificación de mensajería es ínfima en comparación con una cuota mensual de cualquier servicio de seguridad, y dotará al usuario una gran independencia a la hora de vigilar y asegurar sus pertenencias.

Finalmente, con la unión de los dos proyectos [expuestos en el apartado Marco 1.1], a la aplicación móvil se le da un apoyo de almacenamiento mediante un servidor Web, que hace más completa la aplicación, ya que permite tener controlado el historial de sucesos desde cualquier dispositivo que disponga de conexión a Internet, dotando a todo el sistema de una gran flexibilidad, ya que todos sus elementos se pueden considerar móviles. Además de una gran privacidad para el usuario, ya que será sólo él quien tenga acceso a dichos contenidos, consiguiendo un sistema de vigilancia fiable, con elementos cotidianos, fácilmente accesible, barato, privado y personal.

2. Estado del arte

Dentro de este capítulo se sitúa el estudio de las características de los teléfonos móviles actuales, las características de J2ME, y el estado actual de los sistemas de vigilancia.

2.1. Dispositivitos móviles

En este apartado se encuentra una breve noción histórica de la evolución de los teléfonos móviles, y el análisis de las especificaciones y funcionalidades que proporcionan los teléfonos de última generación.

2.1.1. Un poco de historia

El teléfono móvil y la telefonía móvil han evolucionado por el mismo camino, si se encuentra un nuevo servicio que ofrecer, el dispositivo lo ha de soportar y si el terminal es capaz de capturar nuevos tipos de datos, la red debe transportarlos.

En un vistazo histórico, se consideran como los ancestros de los teléfonos móviles, por lo menos, en cuanto a concepto, al radio-teléfono o “handies” de gran difusión durante la segunda guerra mundial.

Figura 1: radio-teléfono o “handie”

La compañía AT&T es la precursora de la investigación de la telefonía móvil, con la intencion de poder situar teléfonos en vehículos automóviles, ante la

imposibilidad de la utilización de un gran número de bandas de radio frecuencia, continuo su investigación hacia un modelo celular.

El primer teléfono móvil, propiamente dicho es desarrollado por Motorola en 1983, el DynaTAC 8000X, con un peso cercano a 1 Kg. y con unas dimensiones de 33 x 4,5 x 9 cm.

Figura 2: DynaTAC 8000X

A partir de este momento, la evolución de los terminales y los servicios se ha categorizado en diferentes generaciones.

La primera generación, 1G, impulsada en 1981 por Ericsson, utilizando canales de radio analógico con modulación FM en un rango de frecuencias superior, que permitían aumentar en número de usuarios. En esos momentos únicamente se permitían las comunicaciones de voz de baja calidad.

La segunda generación, 2G (década de los 90), está apoyada en la digitalización de las comunicaciones, se permite el envío de datos a baja velocidad y voz con cierta calidad, en base a la comunicación GSM.

Si bien, el hecho de enviar datos supone un gran avance, poco a poco comienza a quedarse obsoleto ante la imposibilidad de envío de datos multimedia, y el auge que este tipo comienza a tener en la sociedad.

Por ello, en la conocida como generación 2.5 G, se aumenta la velocidad de transmisión de datos mediante el sistema GPRS, que permite el envío de mensajes con contenido multimedia.

Finalmente, la tercera generación, 3G, es la que se está desarrollando en la actualidad. Está asociada a la comunicación UMTS que permite mayores velocidades de transmisión, por lo que cualquier tipo de contenido multimedia que el teléfono es capaz de capturar se puede transportar por la red. Posibilita la conexión a Internet, por lo que los proveedores de contenidos y servicios adquieren gran relevancia.

En la evolución, se puede observar que, en un primer momento, no se atendió a la evolución de dispositivo, sino a la posibilidad de conexión, por lo que la tecnología de comunicación es más importante mientras que, posteriormente, los contenidos cobran mayor importancia, y es la tecnología la que ha de evolucionar para soportar los formatos que el dispositivo es capaz de reconocer.

En cuanto a la apariencia, con el paso de los años, se ha buscado reducir el tamaño y conseguir el diseño más innovador hasta los actuales móviles táctiles [31].

Figura 3: Evolución de los dispositivos móviles

En el teléfono móvil actual, podemos encontrar diferentes tipos de conexión a Internet, aplicaciones cada vez más complejas, juegos, reproductores de música y video, una agenda totalmente personalizable, hasta el punto, que se suele incluir que el teléfono además de todas estas características, sirve para realizar llamadas y enviar mensajes, lo que es el fin último, pero que no se tiene en consideración a la hora de elegir el nuevo terminal a comprar.

2.1.2. El teléfono móvil actual y sus características

Para el análisis de las capacidades de los terminales actuales, se han escogido los tres terminales con consideración de última generación en el momento actual (otoño 2009), Nokia N97 [17], Iphone [14] y Htc Magic [13], mostrando un breve resumen de las características que se presupone que tiene un teléfono móvil actual, que puede ser ampliado en las especificaciones técnicas de cada terminal, en el cual, también aparecen las características del teléfono móvil en el que se ha comprobado el funcionamiento de la aplicación, Nokia 5800 Express Music [16], y en un guiño histórico, las del teléfono móvil con mayor índice de aceptación a lo largo de la historia de los terminales, el Nokia 3210 [15].

¿Qué se busca en un teléfono a la hora de adquirirlo?. Para responder a esta pregunta, primero hay que atender a los gustos y utilidad que va a darle la persona que se realiza la pregunta.

Se busca un diseño innovador, gran capacidad de memoria, potencia de procesamiento, localización y capacidades de navegación (GPS), facilidad de uso, elementos multimedia, conectividad variable, autonomía y un largo etcétera.

En cuanto al diseño, se busca un terminal ligero, extremadamente delgado y con una gran pantalla (también condicionada por el aumento de la utilización como reproductor multimedia) normalmente táctil. Los botones se están perdiendo, tratando de ocupar ese espacio en a favor de la pantalla.

En cuanto a las capacidades propias del teléfono, la capacidad de procesamiento ha aumentado exponencialmente, con la intención de en algún momento convertirse en auténticos ordenadores, así la memoria interna además de crecer también, en numerosos terminales se puede ampliar introduciendo una tarjeta externa.

Un teléfono, cada vez se convierte más en un gran reproductor multimedia con el que realizar llamadas y enviar mensajes. Así, incorporan cámaras de foto y video cada vez con más prestaciones con capacidad de grabación, y reproductores de audio y vídeo de multitud de formatos.

A parte de la propia comunicación en base al sistema de telefonía, se le añaden otros tipos como la conexión bluetooth para el intercambio de datos en corto alcance de forma gratuita y diferentes formas de conexión a Internet, entre las que destaca el acceso a redes WIFI.

La autonomía, aunque ha aumentado considerablemente, sigue siendo uno de los puntos a reforzar, ya que se buscan cada vez baterías de menor tamaño con mayor capacidad, y en estos momentos, el alto grado de utilización de los recursos supone un gasto de energía muy elevado.

También es importante prestar atención a las aplicaciones que se pueden desarrollar, es destacable una buena agenda personalizable, sistemas de alarmas, conversores, juegos, navegadores WAP, y una infinidad de ellas que dependen de la imaginación de los desarrolladores.

2.2. J2ME

Dentro de este apartado se encuentra una breve descripción de la arquitectura de la plataforma J2ME, el ciclo de vida de una aplicación en J2ME y una breve especificación del modelo de programación. Finalmente se muestran los APIs utilizados para el desarrollo.

2.2.1. Arquitectura

J2ME es una plataforma o conjunto de especificaciones y tecnologías que están pensadas para cubrir las necesidades de los diferentes dispositivos móviles existentes en el mercado. Es importante eliminar la idea de que J2ME únicamente se utiliza para teléfonos móviles, si bien, en este campo tiene su mayor auge y aprovechamiento, cubre muchos más campos. Este hecho es el que hace innecesario que una única solución abarque a todos los dispositivos, puesto que cada uno de ellos posee unas características que para otro pueden ser irrelevantes o incluso inexistentes.

Es un lenguaje de programación derivado de java, pero muy limitado. La versión completa del lenguaje de programación sería demasiado pesada para un dispositivo de memoria limitada, por lo que trata de incluir las funcionalidades básicas para cada dispositivo, en busca de un paquete que permita una programación básica con baja ocupación.

Para distinguir entre los diferentes dispositivos, existen varias configuraciones. Cada una de ellas es un especificación que detalla una maquina virtual y el conjunto mínimo de clases java que debe incluir para un grupo de dispositivos.

Configuración CDC (*Connected Device Configuration*)

Está orientada a dispositivos con mayor capacidad computacional y de memoria, como pueden ser los decodificadores de TV, sistemas de navegación o incluso electrodomésticos. En todos ellos la interfaz de usuario es limitada, por lo que no es necesario dotar de grandes capacidades a este aspecto. Se les supone una cierta

conectividad a algún tipo de red (no solo tiene que ser a Internet), y que disponen de un soporte completo a una maquina virtual java (JVM).

Configuración CLDC (*Connected Limited Device Configuration*)

Está orientada a dispositivos con capacidad de conexión a cualquier red y con limitaciones de computación y memoria, y también de interfaz gráfica, pero con la necesidad de tenerla de forma mucho más completa que en CDC, también tiene una limitación externa a la programación pero que está asociada al procesamiento interno, que es la escasa autonomía de las baterías.

Este es el caso de los teléfonos móviles, y por tanto la configuración utilizada será CLDC, que posee unos paquetes básicos:

Paquete	Descripción
Java.io	Encargada de la gestión de Entrada/Salida.
Java.lang	Maquina virtual
Java.util	Utilidades estándar
Javax.microedition.io	Conexión

Tabla 1: Paquetes configuración CLDC

El conjunto mínimo de clases que incluyen la configuración no es suficiente para realizar la programación de una aplicación en un dispositivo móvil, por lo que es necesario completarlo con un perfil.

El Perfil define el modelo de ciclo de vida de la aplicación, el interfaz de usuario y el acceso a las propiedades específicas del dispositivo

Para nuestro caso, es necesario centrarnos en el perfil MIDP 2.0, el cual en su especificación contiene, clases para el control de la interfaz gráfica, el acceso a la memoria persistente del dispositivo (RMS), el ciclo de vida del midlet, gestión de las conexiones de red, un pequeño grupo de acciones para la reproducción de audio, certificación y seguridad en las conexiones y el resto de paquetes propios de la configuración CLDC [ver Tabla 1] [1].

Además, existen numerosos APIs que permiten el control de elementos que no se encuentran en todos los dispositivos. Para nuestro caso, deberemos acceder al sistema de almacenamiento interno del móvil (incluido en JSR 118, RMS [3]), a los dispositivos y capacidades multimedia (JSR 135 MMAPI [2]), enviar mensajes de un dispositivo a otro (JSR 205 WMA [4]), bluetooth (JSR 082 [6]), conectarnos a Internet (incluido en JSR 118, javax.mocroedition.io [1]), y finalmente utilizar un nuevo motor gráfico para la presentación (LWUIT).

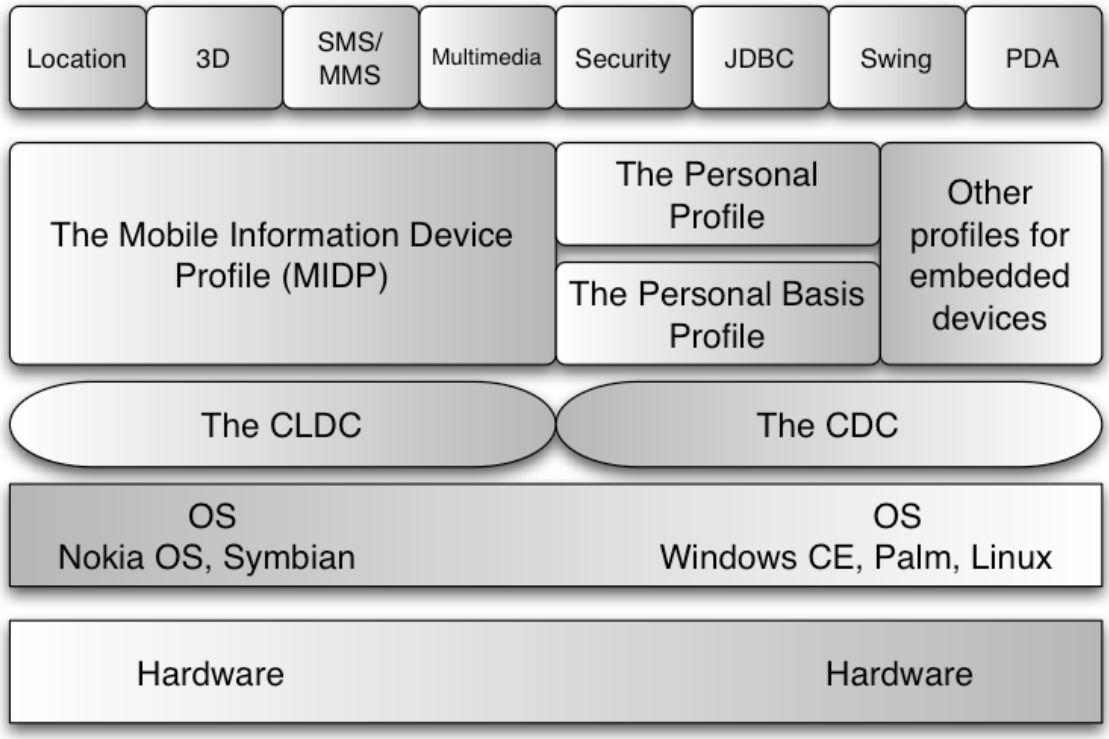

Figura 4: Arquitectura J2ME

La arquitectura de la plataforma, además de la configuración y el perfil, tiene asociada una maquina virtual que es la encargada de interpretar y traducir el programa java compilado (*byteCode*) a órdenes que entiende el sistema propio del dispositivo (código maquina).

La JVM (*java virtual machine*) tradicional de java es demasiado pesada para un dispositivo con memoria reducida, por ello, dentro de J2ME tenemos dos máquinas virtuales principales de referencia: CVM, que está asociada a la configuración CDC, y por tanto a los dispositivos que en ella se especifican. Por otro lado, KVM está

diseñada desde un inicio, sin basarse en las ya existentes, para centrarse en dispositivos que cumplen los requisitos de la configuración CLCD, por tanto está preparada para dispositivos con poca memoria, capacidad de proceso limitado y con conexión a cualquier tipo de red de manera intermitente e inalámbrica. Posee un tamaño reducido (40-80KB) que le permite ser bastante completa y rápida, además de modulable y muy portable.

2.2.2. Midlet

Un midlet es una aplicación Java que se desarrolla con J2ME, siguiendo las características que le indica una configuración y utilizando las capacidades que se indican en los perfiles.

Para que se puedan cumplir todas las fases del ciclo de vida de una aplicación es necesaria la actuación de un sistema de gestión de aplicaciones (AMS), el cual es propio de cada dispositivo.

Dentro del ciclo de vida de la aplicación o midlet, en un vistazo global hay que distinguir dos grandes apartados: el diseño y la utilización.

En la fase de diseño, participa el programador y las herramientas que utiliza para ello, mientras que la fase de utilización es responsabilidad del sistema AMS.

Dentro del diseño, se atiende a dos acciones, la creación (programación pura) y la publicación (subida al servidor y descarga por parte del dispositivo).

En la fase de creación, es totalmente necesario dejar perfectamente aclarados los estados por los que pasa la aplicación. Las características de estos estados estarán incluidas en los métodos del ciclo de vida del midlet [ver Apartado 2.2.3.1].

Una aplicación, se empaqueta en un archivo extensión .jar, el cual además del código contendrá el manifiesto en donde se encuentran los datos de los archivos que están incluidos. Y un fichero extensión .jad, donde se encuentran las características de la aplicación, y será necesario para que el dispositivo compruebe la compatibilidad de la aplicación con el terminal.

Las responsabilidades de la AMS, están relacionadas con los procesos de descarga e instalación, la utilización (atendiendo a los estados del midlet), la actualización y el borrado.

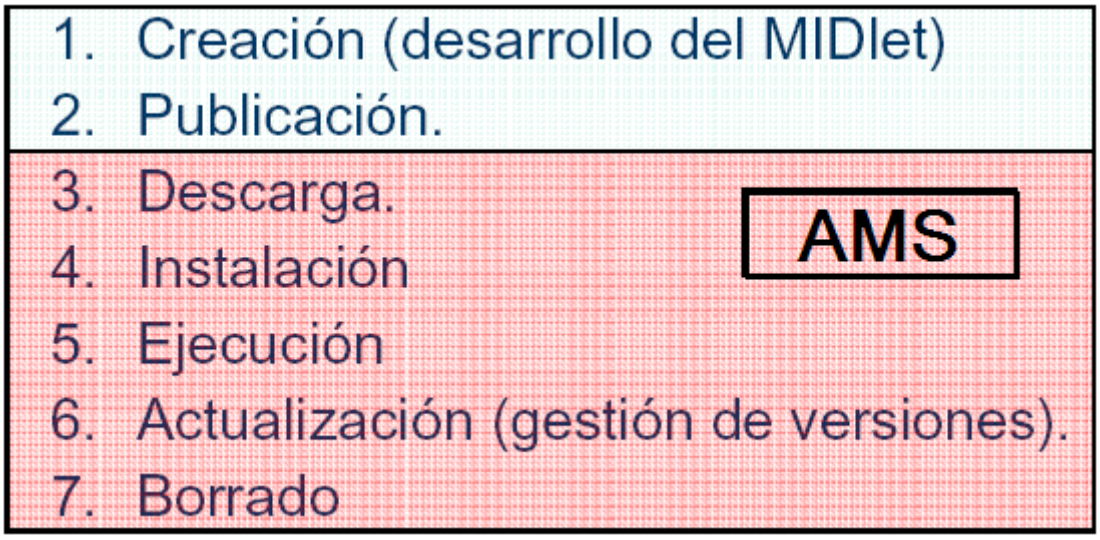

Figura 5: Ciclo de vida de una aplicación

2.2.3. Programación

Dentro del proceso de programación de una aplicación, no solo hay que atender a la escritura del código, por lo que este apartado, además de la codificación, muestra las características a tener en cuenta a la hora de realizar una aplicación J2ME.

2.2.3.1. Ciclo de vida

Un midlet, debe siempre contener los métodos del ciclo de vida. La implementación de estos métodos es condición necesaria para el funcionamiento de la aplicación.

Cuando se crea una aplicación, tras pasar por el constructor (en caso de que exista), la siguiente acción será la llamada al método startApp(). Con él se crea el hilo principal de ejecución, que estará siempre activo a lo largo de la aplicación, salvo que alteremos el estado de la misma.

Dichas alteraciones del estado, son causadas por los otros dos métodos de gestión del ciclo de vida: pauseApp(), y destroyApp().

Mediante el método de pausa, paramos la ejecución de la aplicación pudiendo reemprenderla con la llamada al método startApp(). Mientras que si utilizamos el método destroyApp(), la aplicación se cerrará por completo.

Los métodos del ciclo de vida, además de tener su función, se pueden programar acciones a realizar cuando se utilizan.

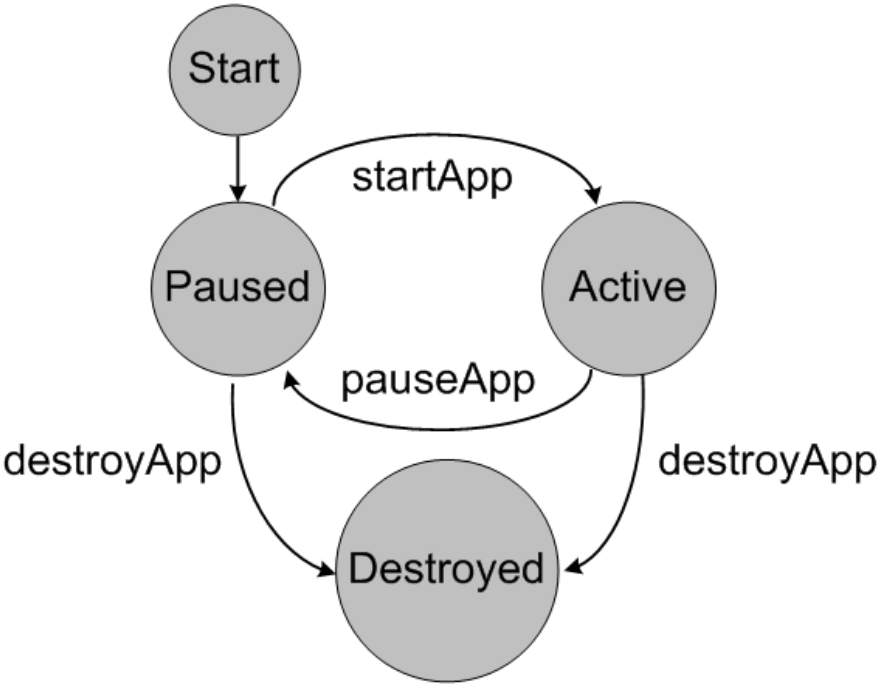

Figura 6: Ciclo de vida de un midlet

2.2.3.2. Gestión de eventos

Dentro de una aplicación, se generan numerosos eventos, tanto internos como externos que completan la funcionalidad. En un dispositivo móvil, los eventos son de diversa naturaleza: provocados por el usuario (al pulsar teclas o botones) o externos (llegada de mensajes, respuestas de servidores). Y todos ellos deben ser capturados si se desea que modifiquen el funcionamiento general de la aplicación.

Cada API, proporciona los métodos encargados de ello, pudiendo incluir las acciones a realizar cuando se producen. Los de bajo nivel se encargaran de los eventos provocados por la acción de pulsar una tecla, o en caso de que el terminal sea táctil, la presión sobre una determinada parte de la pantalla. Mientras que los eventos provocados por elementos programados (comandos, botones, listas, etc.), son controlados por los APIs de alto nivel.

Cada pantalla de la aplicación tendrá un escuchador diferente. Si se desea que sea el mismo para toda la aplicación, se debe crear (es indiferente el lugar) y registrar el escuchador en cada pantalla.

Existe una diferenciación entre los elementos y sus escuchadores, por tanto, la interfaz ActionListener gestionara los eventos por acciones internas provocadas por comandos o botones, es decir, acciones de alto nivel, mientras que para la gestión de las teclas se puede utilizar los métodos keyPressed(al pulsar), keyReleased(al soltar), keyRepeat(al repetir), se trata de la acción de bajo nivel.

Cada componente puede realizar una acción diferente para un mismo evento, por ejemplo, reaccionar de manera diferente ante la pulsación de una tecla. Por tanto, es posible registrar escuchadores particulares a los componentes, o incluirlo en un escuchador general, en el que se atiende primero a la procedencia del evento y posteriormente se indica la acción a realizar.

Además, existen otro tipo de escuchadores que son más específicos a las características de cada componente, por ejemplo, si cambiamos de selección en una lista o si cambia el contenido o el estilo. Todos estos escuchadores, así como su funcionamiento, están incluidos en las diferentes especificaciones de los componentes.

2.2.3.3. Interfaz gráfica

La propia especificación del perfil Midp 2.0 proporciona una serie de clases y métodos asociados que se encargan del control de la pantalla y todos sus elementos gráficos.

Se distingue entre APIs de bajo nivel (basados principalmente en la clase canvas) y APIs de alto nivel (basados principalmente en la clase Display).

El API de bajo nivel realiza un control de la pantalla a nivel de píxel, lo que permite realizar un diseño de elementos complejos, pero requiere un gran trabajo para llegar a las formas deseadas. Se puede le puede dar a cada píxel la apariencia que se quiera. Toda la manipulación de los pixeles está incluida dentro del método paint(), que es obligatorio implementarlo al utilizar un objeto canvas. Esta flexibilidad en el pintado y diseño de la pantalla, supone, por el contra una menor portabilidad, un diseño posiblemente funcionará tal y como se desea, en el dispositivo sobre el que se programa, no en todos ellos. Es muy utilizado en el diseño de juegos.

El API de alto nivel, se puede considerar que define los objetos o elementos de uso habitual en una aplicación. Son elementos en esencia inalterables, de los cuales si se pueden cambiar alguna de sus características pero su morfología es bastante cerrada e inalterable. Esto permite una gran portabilidad en los dispositivos, puesto que las características básicas son iguales para todos los dispositivos, pero da muy poca flexibilidad en el diseño “artístico” de la aplicación.

Ambos APIs no son excluyentes dentro de una aplicación, pero sí lo son dentro de una pantalla, si se trabaja a nivel de canvas, ha de ser en toda la pantalla, así como si se elige utilizar un objeto Display, este gobernará toda la pantalla.

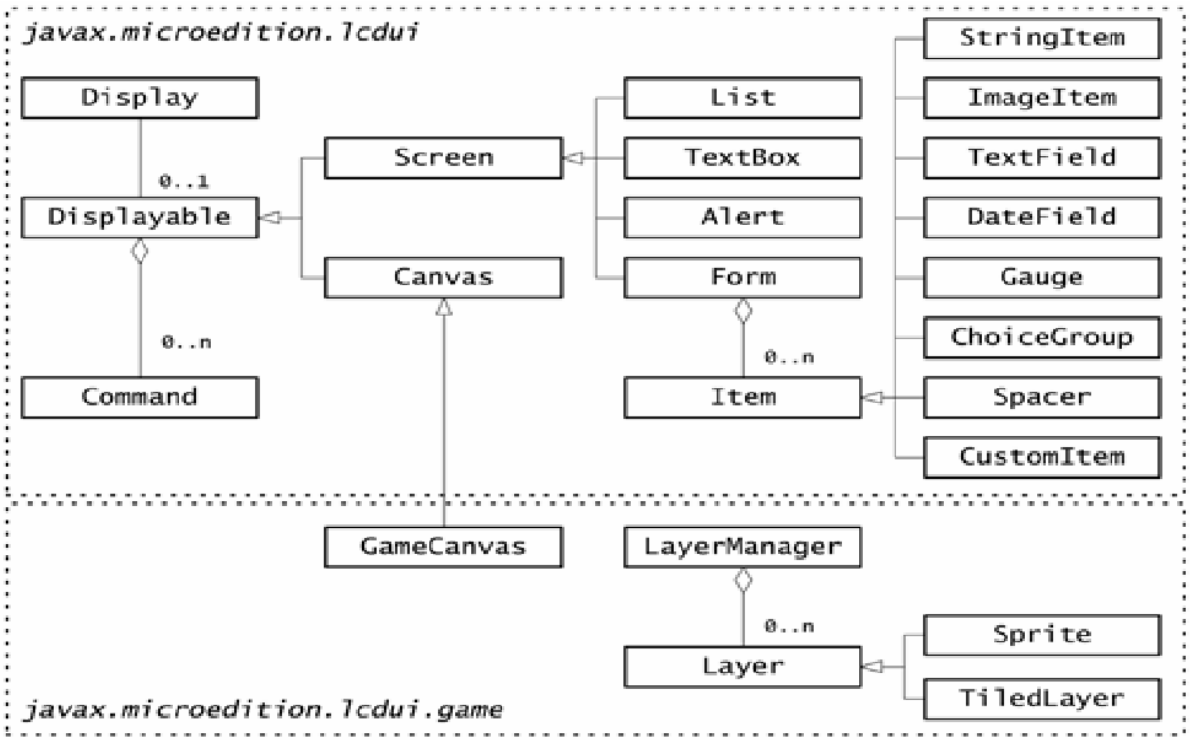

Figura 7: Paquetes para la interfaz gráfica

Esta implementación de la interfaz gráfica de una aplicación, no permite realizar una separación entre los elementos gráficos y la funcionalidad de la aplicación, ya que cada acción, además de estar asociado al elemento que lo produce, también lo está a la pantalla en la que está situado.

También es destacable, la utilización de la clase Alert, la cual debe estar relacionada con un objeto displayable para su visualización, lo que hace complicada la programación de alertas no asociadas a ninguna pantalla, es decir, que respondan a eventos globales que pueden producirse en cualquier momento.

Además para conseguir una apariencia visual compleja, el esfuerzo en la programación es muy elevado, y los resultados son poco atractivos, no se aleja nunca del aspecto de aplicación para teléfono móvil.

Por estos últimos casos, se ha decidido utilizar un nuevo motor grafico, aparecido en Julio de 2008, que además de incluir nuevas mejoras y elementos más completos, separa por completo la funcionalidad de la vista, siguiendo por tanto, el MVC (Modelo Vista Controlador), LWUIT [explicado en el apartado 2.2.4.6].

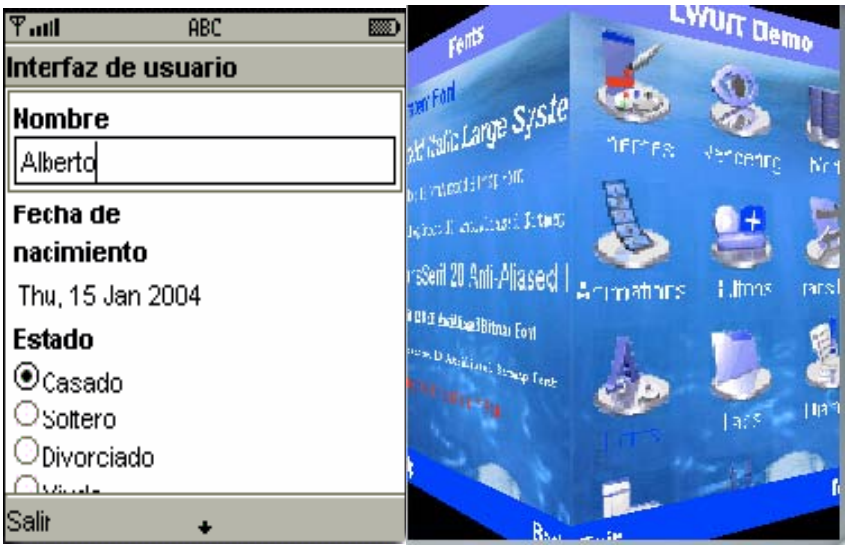

Figura 8: lcdui vs lwuit

2.2.4. APIs utilizados

Dentro de este apartado aparece una breve descripción de los APIs utilizados que incluyen la definición y características básicas, así como la utilización dentro de la aplicación.

2.2.4.1. RMS (Record Management System)

Se trata del sistema de almacenamiento persistente en la memoria del dispositivo. En ningún momento se indica la forma o maneara en la que se almacena dentro del terminal, esta lógica Será diferente para cada implementación. Que un dato se almacene de manera persistente indica que, se mantendrá inalterable entre diferentes ejecuciones de una misma aplicación, salvo que se modifiquen dentro de las mismas, y existirán mientras que estén instaladas las aplicaciones en el dispositivo, cuando se borre la aplicación, también se eliminaran sus datos asociados en la RMS.

Un elemento almacenado se conoce como Record (registro), que se puede combinar con varios de ellos formando un RecordStore.

El RecordStore, es por tanto una colección de 0 o más registros. Dentro de una misma aplicación, no pueden coexistir dos RecordStore con el mismo

nombre y el acceso está basado en la definición de permisos, no es habitual, pero una aplicación puede acceder a cualquier RecordStore de otra aplicación si tiene permiso para ello.

Cada registro (Record), debe estar incluido dentro de un RecordStore. El tipo de datos que se guardan en ellos es el byte [] por lo que, cualquier tipo que se pueda transformar a un array de bytes, se podrá almacenar, realizando la conversión previa al almacenamiento. Cada registro tiene un identificador (recordId) de tipo int, mediante el cual se accede a él para obtener o modificar los datos, y para eliminarlo. La interfaz RecordEnumeration se utiliza para navegar por el RecordStore y acceder a sus registros para el caso en el que no se conozcan sus identificadores, ya que cuando se elimina algún registro, los identificadores dejan de ser consecutivos, y es sumamente complicado seguir la pista a los registros existentes mediante su identificador.

La especificación completa de todas las clases e interfaces que controlan el sistema de almacenamiento persistente se encuentra dentro del paquete javax.microedition.rms [3].

Dentro de la aplicación a desarrollar, se utilizara este sistema para el almacenamiento de las palabras claves y los números propios y de envío.

2.2.4.2. MMAPI (Multimedia API)

Se trata de un API que complementa al disponible en Midp 2.0 para el manejo de audio, de hecho, este es un subconjunto del API completo.

El api completo, proporciona soporte para las nuevas generaciones de reproducción y grabado, no solo de audio, sino que de todos los elementos multimedia que un dispositivo móvil es capaz de capturar y procesar.

Para el procesado multimedia el API se apoya en cuatro elementos fundamentales:

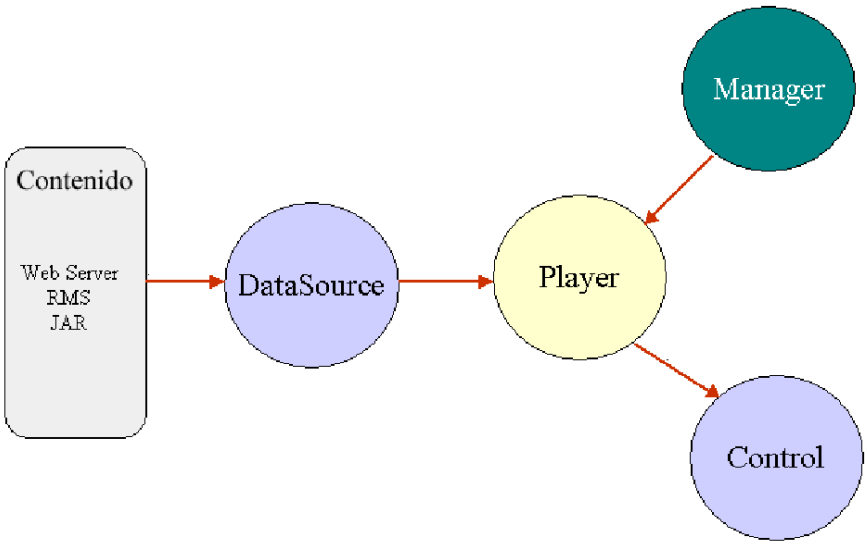

Figura 9: Elementos api multimedia

DataSource: Oculta la forma o manera de leer los datos de la fuente, encapsulando los datos y abstrayéndolos del protocolo utilizado; sirviendo un formato de datos que es capaz de gestionar un player.

Player: Realiza la lectura, proceso y control de la reproducción de los datos que le llegan. Su finalidad es enviar dichos datos a un dispositivo de salida (visual o auditivo).

En su ciclo de vida, pasa por cinco estados, es importante tener consciencia del estado del player, puesto que no todas las acciones que realiza se pueden realizar en todos los estados.

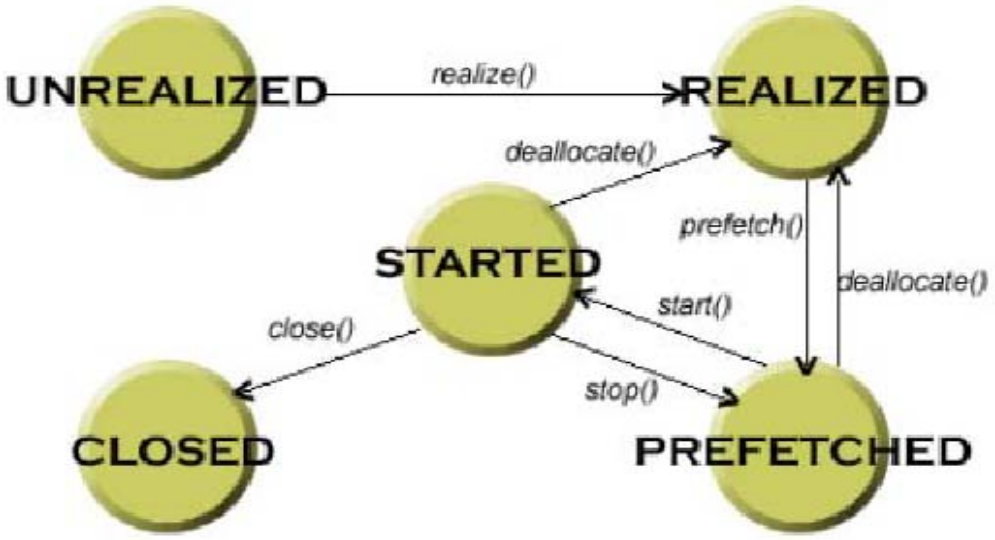

Figura 10: Estados de un player

Unrealized: Estado en el que se encuentra tras ser creado.

Realized: En este estado ya se dispone de la información para adquirir los recursos multimedia

Prefetched: Se establece la conexión para acceder a los datos.

Started: En este estado, ya se pueden procesar los datos, es decir, principalmente se pueden reproducir y modificar sus características.

Closed: Se cierra el player y para volver a utilizarlo hay que crearlo de nuevo.

Manager: Es la clase que se encarga de la creación del player, normalmente en base a un url donde está alojado el contenido o a través de un objeto InputStream indicando el tipo de este. También permite la reproducción instantánea de sonidos propios del teléfono.

Control: Se encarga de obtener los diferentes controles del medio, para permitir modificar sus atributos (volumen, por ejemplo) y obtener las características de visualización (VideoControl) o grabación (RecordControl).

La especificación completa se encuentra dentro de la JSR 135 que contiene los paquetes javax.microedition.media, javax.microedition.media.control, javax.microedition.media.protocol [2].

Dentro de la aplicación este API, es utilizado para el acceso a la cámara, y para tomar fotos de la misma. Para ello, necesitamos un player específico que capture imágenes o vídeo (según las características del dispositivo [ver Tabla 5]) y obtener dentro de los controles, el VideoControl que será el que proporcionará la captación de fotos (getSnapShot()). Además, este control, también proporciona las características visuales, pero en este caso no se utilizan ya que la visualización corresponde a un objeto MediaComponent, incluido en el nuevo motor grafico LWUIT.

2.2.4.3. WMA (Wireless Message Api)

Es el API que se encarga de especificar las clases e interfaces para el envío y recepción de mensajes (SMS o MMS) entre dispositivos.

Para la creación de mensajes de texto, hay que atender al paquete javax.wireless.messaging.TextMessage, mientras que para los mensajes multimedia se atiende al paquete javax.wireless.messaging.MultipartMessage, si bien también existe la extensión BynaryMessage, no tiene la suficiente capacidad para el envío de imágenes de gran tamaño.

También hay que diferenciar el funcionamiento para los casos de recepción y envío, ya que se utilizará el modo servidor para la recepción (aunque también permita el envío) y el modo cliente, que solo permite el envío de mensajes a un destinatario, esto estará especificado mediante el uso del paquete javax.wireless.messaging.MessageConection.

Para la recepción se puede optar por dos estrategias, de forma síncrona o de manera asíncrona. Para este proyecto, es necesaria la recepción de forma asíncrona, por lo que hay que atender al interfaz MessageListener().

La última revisión de este API es la versión 1.0.2, correspondiente a la JSR 205 [4].

Dentro de la aplicación, los mensajes de texto se utilizan para el control remoto de la aplicación de captura y detección, por parte de la aplicación de recepción; tanto para la funcionalidad como para la configuración. Mientras que los mensajes multimedia se utilizan para el envío de la imagen en la que se ha detectado movimiento.

2.2.4.4. BLUETOOTH

Es el API encargado de especificar las clases e interfaces para el intercambio de datos mediante una comunicación bluetooth. Las características principales de esta son pequeño tamaño, consumo moderado y en caso de los terminales móviles, gratuito.

Los procesos de bajo nivel que hay que seguir para realizar una conexión mediante esta tecnología, son demasiados complejos, y pueden provocar la pérdida de la visión de desarrollo, con la intención de dotar de facilidad a la implementación. Este campo, es el que cubre esta especificación, mediante sencillos métodos, esconde la complejidad de dichos procesos.

Permite trabajar tanto con aplicaciones nativas como con la especificación javaBluetooth, por lo que adquiere una gran portabilidad entre dispositivos, ya que no está sujeto a la forma de utilización sino a las características que ha de tener cualquier comunicación.

Este API, proporciona las clases e interfaces que controlan el registro de servicios, el descubrimiento de dispositivos y los propios servicios, recepción y envío de datos (voz no soportada) y ofrecer seguridad a todas estas actividades.

Se distingue entre dos modos de funcionamiento, servidor y cliente. Dentro del servidor se crean y registran los servicios, mientras que el cliente es el que trata de descubrir los servicios e inicia la comunicación de datos.

Toda la especificación se encuentra dentro de la JSR 82 [6], que comprende los paquetes javax.bluetooth y javax.obex. En este caso solo es necesario el paquete bluetooth, puesto que obex es un protocolo de intercambio de objetos que no se utilizar en este proyecto.

La aplicación, utiliza este API para su finalización de manera remota mediante el sistema de comunicación bluetooth.

2.2.4.5. CONEXIÓN

No se trata de un API propiamente dicho, ya que se encuentra incluido dentro de la especificación básica de CLDC en lo que a conexiones se refiere.

Dentro de los múltiples tipos de conexión, este apartado se centra en la conexión a Internet mediante el protocolo http.

Por tanto, en vez de analizar un API, se analiza una interfaz encargada de establecer los métodos y constantes necesarias para la comunicación.

Existen métodos, que controlan las posibles fases del protocolo http, por tanto hay métodos de configuración, de conexión y de cierre.

Dentro de la configuración se deberá establecer el tipo de conexión a realizar, bien sea de tipo Get o de tipo Post. En todo caso, se debe leer la respuesta y en caso satisfactorio se pueden obtener los datos de la comunicación.

Esta interfaz se encuentra dentro de la JSR 118 [1] dentro del paquete javax.microedition.io.

En la aplicación, la conexión a través del protocolo http se utiliza en dos casos: Comprobar que el usuario esta dentro de la base de datos, y para el envío de la imagen en la que se ha detectado movimiento.

2.2.4.6. LWUIT

Se trata de un nuevo motor grafico (Julio 2008), que mejora notablemente el aspecto visual que proporciona el paquete javax.microedition.lcdui, ampliando notablemente su funcionalidad y aportando nuevos elementos como son ComboBox(listas con una única selección), TabbedPane (pestañas), Dialog (mejora sustancial de Alert) y Calendar (calendario propio). Incluye también todos los layouts (disposición de elementos en la pantalla [ver Figura 14]), que provee el AWT (motor gráfico de Java).

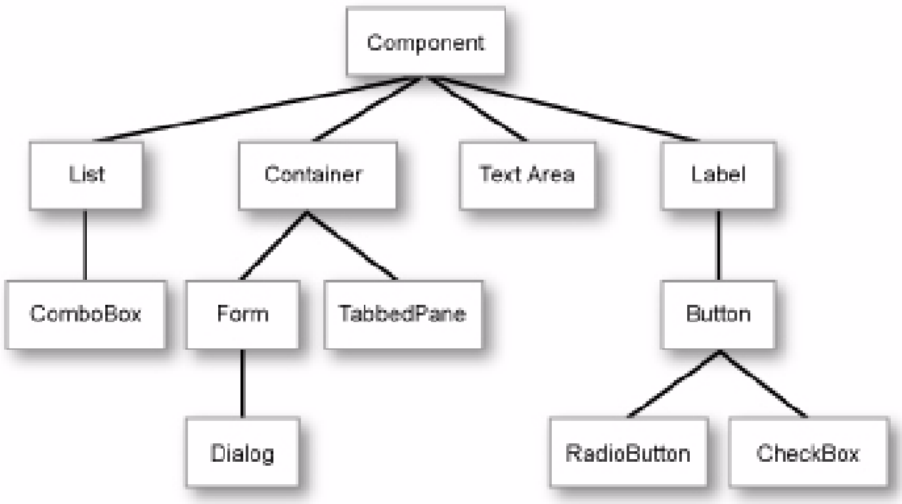

Figura 11: Componentes LWUIT

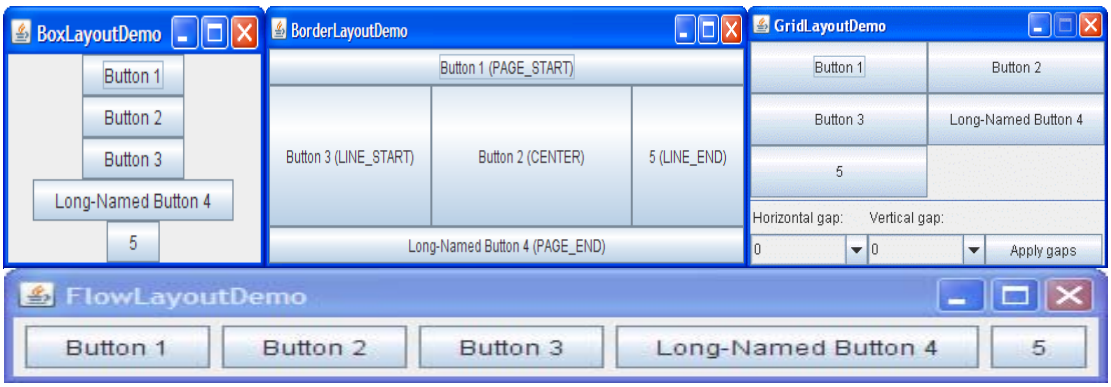

Figura 12: Layouts

Un aspecto muy importante, es que permite seguir el modelo MVC (vista-controlador), separando la parte visual (colocación y estilo) del flujo de la aplicación, ya que permite registrar los eventos que se quieren utilizar en el propio componente.

Está preparado para dispositivos táctiles, permitiendo la programación de eventos referidos a la pulsación de cualquier punto de la pantalla (pulsar, soltar y mantener), y además, una aplicación desarrollada para un dispositivo con teclado, funciona en uno táctil sin que haya que realizar ninguna otra acción para hacerlo compatible.

Permite la utilización de animaciones en 2D y 3D, especialmente para las transiciones (pasos de una pantalla a otra), de una forma sencilla en la que el programador solo se debe preocupar de que tipo, velocidad y desplazamiento desea, y con esos tres parámetros es el motor grafico el que crea el efecto.

En cuanto al uso de componentes, es mucho más abierto y se puede personalizar casi hasta donde llegue la imaginación del diseñador, y sin tener que recurrir al pintado manual. Esto es debido a la inclusión de Renderer y Model, con los cuales se puede controlar el aspecto que puede tener una lista, por ejemplo, y dictaminar que aparece cuando está seleccionado, cuando se navega, o cuando se deselecciona. Además, permite modificar el estilo de cada componente de manera sencilla, accediendo al objeto Style de cada uno de ellos, pudiendo variar los colores, las fuentes, los bordes…

Proporciona además un editor de recursos, con una interfaz bastante sencilla e intuitiva, que permite seleccionar los componentes y añadirle las características que se deseen, bien sea la fuente, la imagen de fondo, los colores de fondo, de selección etc. Creando un archivo de recursos (.res), de fácil acceso, que facilita la programación y sobre todo la hace mucho más limpia. Se podría asemejar un archivo .res a una hoja de estilo CSS, con la particularidad que mediante programación se puede sobrescribir el estilo de un determinado componente, sin afectar al resto.

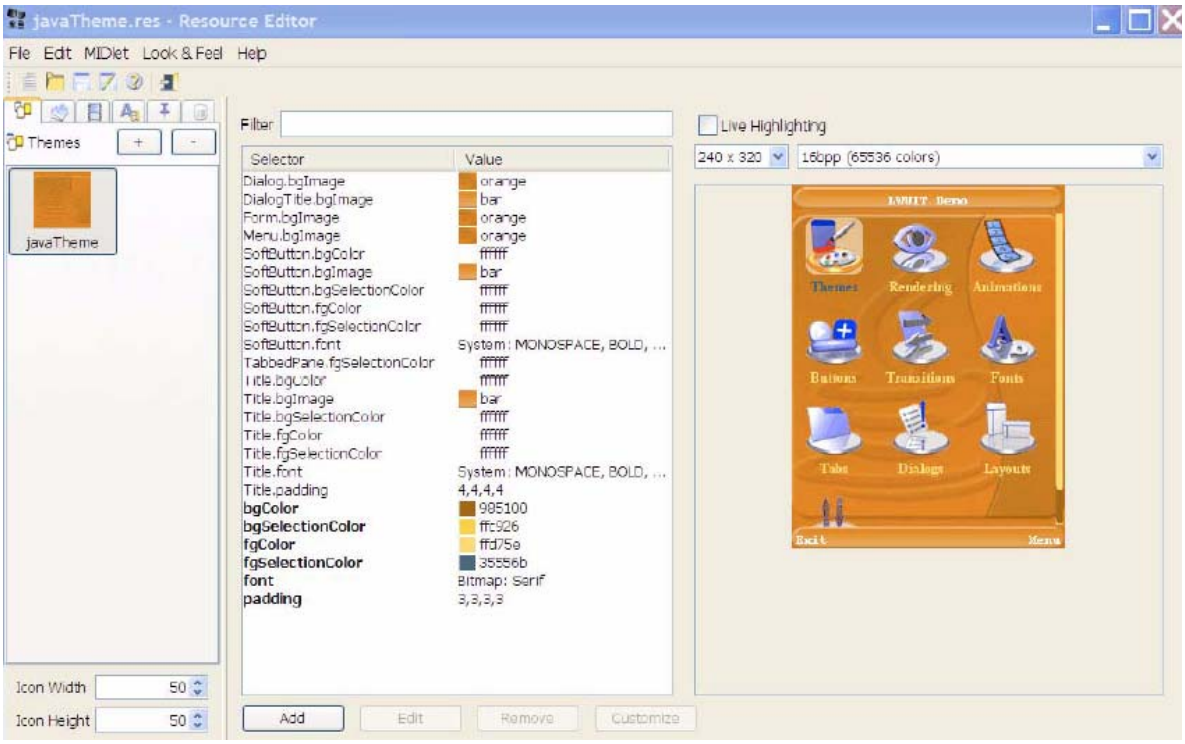

Figura 13: LWUIT ResourceEdit

Para un programador de Swing y AWT, no supone ningún tipo de problema utilizar esta API, lo que abre el mundo del diseño móvil en J2ME, ya que posee una buena documentación, bastante similar a estos modelos.

Su licencia es libre, la mayoría de terminales actuales permiten su utilización y lo llevan incluido en las especificaciones propias, y su peso, tras un proceso de ofuscación[1], puede reducirse notablemente. A su vez requiere que los teléfonos dispongan de una configuración CLDC 1.1 o superior y De MIDP 2.0. La

[1] *Proceso mediante el cual se elimina el código no usado dentro de una aplicación y además se encripta con la intención de evitar la copia de alguna de las partes.*

última versión estable es la 1_2 (julio 2009), mientras que este proyecto ha sido desarrollado con la versión de julio de 2008 [18].

Puesto que una imagen vale más que mil palabras, para ver las capacidades de este nuevo motor grafico se recomienda acceder al vídeo de referencia, así como, la aplicación de prueba proporcionada con la librería[31][36].

2.3. Sistemas de seguridad

Dentro de este apartado se encuentra una relación historia de la evolución de los sistemas de seguridad hasta el análisis de los sistemas de moda en la actualidad.

2.3.1. Un poco de historia

La seguridad es una de las grandes preocupaciones de la sociedad actual, aunque por sistema de seguridad entendamos un sofisticado conjunto de elementos hardware y software supervisados por una persona o grupo de personas, la historia de los sistemas de seguridad no empieza con la aparición de cualquier tecnología.

Un sistema de seguridad, es cualquier elemento que garantice la "Fianza u obligación de indemnidad a favor de alguien, regularmente en materia de intereses" [10], que otorgue confianza al usuario, y que no produzca ningún tipo de molestia. Por lo tanto, para hablar del primer sistema de seguridad, se ha de referenciar a la figura del guarda de seguridad. Durante mucho tiempo, e incluso en la actualidad, se considera a los animales, concretamente a los perros, como guardianes, por lo tanto, tampoco es descabellado considerarlos como un sistema de seguridad.

Figura 14: Sereno, seguridad pública

Para hablar de evolución de sistemas de seguridad, ya si, asociados a las innovaciones tecnológicas, conviene realizar una distinción en tres generaciones, las cuales, al avanzar implican una mayor complejidad relacionada con las necesidades que puede requerir el usuario.

La primera generación comprende los sistemas de seguridad cuya finalidad es detectar una situación anómala y realizar un aviso, con un pequeño sistema de control para dictaminar cuando debe estar activo. Son los dispositivos de tipo alarma sonora, muy utilizados como medida disuasoria.

La segunda generación consiste en dispositivos capaces de controlar eventos o situaciones y tomar decisiones en función de ellos. Elimina la fuerte dependencia con el usuario final al no necesitar su participación para el inicio o fin, y poseen cierta capacidad para discernir entre situaciones “alarmantes” y situaciones poco usuales. Se trata de sistemas de activación por cierre, por ejemplo sensores de movimiento en salas.

La tercera generación, incluye elementos de monitorización sin obligar al usuario o responsable a estar en la misma localización del lugar que se está vigilando. Permite una mayor flexibilidad, y un aumento en el número de zonas a controlar. Además, con un sistema de almacenamiento asociado, dan soporte para realizar un historial que puede utilizarse, entre otras funciones, para automatizar el sistema y dictaminar cuales son o no situaciones anómalas en función de los datos almacenados. Se trata de sistemas de grabación, por ejemplo CCTV (Circuito cerrado de televisión).

Figura 15: Generaciones de sistemas de seguridad

Dentro de estas tres generaciones, los sistemas encargados del "Cuidado y atención exacta en las cosas que están a cargo de cada uno" [10], se encuentran dentro de la tercera generación. Por lo que el estudio de este mercado será el adecuado para situar la competencia de esta aplicación.

2.3.2. El sistema actual

Un sistema de televigilancia actual, consta de un conjunto de cámaras de vídeo, interconectadas con un servidor o monitor (mediante cualquier tipo de protocolo), que permite la visualización de las imágenes y, en caso de tener un dispositivo de almacenamiento, guardarlo a modo de historial.

Este sistema se conoce como CCTV (*Closed Circuit Television, Circuito Cerrado de Televisión*). La consideración de cerrado, viene de que al contrario de los sistemas de televisión de difusión todos sus elementos están enlazados, y el número de espectadores es muy limitado (usuario o responsable de control de monitores).

El número de cámaras depende del emplazamiento a vigilar y del detalle con el que se quiera realizar la vigilancia. También es muy importante tener en cuenta las características de la cámara y las condiciones en las que se va a situar. Se debe atender a parámetros de luminosidad, capacidad y variabilidad angular y disposición de las mismas. Puesto que para vigilancia de zonas con poca luz o en horario nocturno, necesitara tener la capacidad de "visión nocturna". Las características de captación

angular no podrán ser las mismas para una cámara situada en una sala de grandes dimensiones, que para una pequeña sala. La disposición debe estudiarse, y en caso necesario, utilizar cámaras con sistemas de movimiento que permita variar el ángulo de visión, así como el zoom con el que se toman las imágenes.

Unida a la captación, las imágenes pueden ser tratadas en tiempo real mediante el apoyo de un sistema informático, lo que hace crecer las prestaciones de manera exponencial. Usando algoritmos de detección de movimiento, reconocimiento de imágenes y seguimiento de personas u objetos. Con este último apartado, se avanza hacia una automatización cada vez más pronunciada del servicio.

En cuanto a la visión y campo de negocio, las ofertas no varían en el concepto de circuito cerrado de televisión, es decir, un sistema de cámaras con soporte de red y monitorización. Pero si existe un gran abanico de posibilidades en cuanto a su implementación real del sistema, es decir, el tipo de cámara, el protocolo de transporte y la monitorización final, además de los servicios añadidos desde un servidor.

Sobre el tipo de cámaras, se pueden realizar varias distinciones en cuanto a color: banco y negro, color propiamente dicho o visión nocturna (no específicamente un color); movilidad: fija, con movimiento, o con capacidad de rotación (el famoso domo); situación: de interior o exterior; protocolo de comunicación: cámaras IP; inalámbricas o con cableado; y tamaño: cámaras espías.

Figura 16: Cámaras de vigilancia

El tipo de protocolo de red, viene dado por el tipo de cámara y su protocolo de comunicación. Además es importante la estructura arquitectónica en la que se va a instalar, siempre será más sencillo realizar un estudio de cableado en un edificio de nueva construcción que planificar sobre una estructura ya hecha, por ello, los sistemas inalámbricos y por IP, están cobrando mayor importancia en la actualidad. Ya que estos sistemas permiten una flexibilidad infinitamente mayor que el cable tradicional, aunando portabilidad y fiabilidad.

El sistema de monitorización es el elemento final de este tipo de sistemas, puede abarcar desde una simple pantalla a cargo de una persona responsable de dictaminar cuando se produce alguna situación anómala, hasta potentes servidores con servicios adicionales de refuerzo, capaces de realizar la detección por sí mismos. Al incluir un ordenador como parte del proceso, se le añade la gran capacidad de transmisión que proporciona Internet, y por tanto, permite la distribución de las imágenes a cualquier terminal que disponga de conexión.

Esta aplicación cumpliría la idea de circuito cerrado, puesto que los destinatarios son limitados, y seguiría la filosofía de automatización del servicio, con la incorporación de detección de movimiento, en este caso, existirá una cámara, la del teléfono, que puede suponer la mayor diferencia frente a los sistemas con cámaras de mayores prestaciones.

El problema de este tipo de sistemas, sin duda, es el elevado precio debido al tipo de tecnología que se utiliza, cuantas más prestaciones, más sofisticados son sus elementos, y mayor coste tiene el equipo. Además del precio de mantenimiento y de supervisión por parte de una tercera empresa, en caso de solicitar ese tipo de servicio.

Los precios para sistemas oscilan entre los 300 € para sistemas con dos cámaras y grabador, hasta los 2600€ de sistemas con cámaras IP, y servidor asociado [28] [33], sin incluir los servicios de instalación, mantenimiento y vigilancia asistida.

“No existen problemas iguales en Seguridad y por tanto, tampoco existen soluciones iguales” [8]. Por lo que, para cada caso, se estudia la situación, y se realiza

un presupuesto. Este presupuesto, depende de los elementos que se utilicen. Se trata de buscar soluciones para localizaciones de gran entidad: edificios, negocios, aparcamientos, organismos públicos…. Siendo quizá excesivo montar un CCTV en un hogar (de dimensiones modestas). Aunque, últimamente, si que están cobrando cierta fama, tanto en negocios, como en edificios de viviendas (zonas comunes) y hasta en las propias calles de las ciudades.

La utilización de este tipo de sistemas en lugares públicos, se debe acoger a Ley de protección de datos, y además, a la percepción popular de sentirse observado, por lo que se vierten opiniones en diversas corrientes, en la lucha entre ser observado por seguridad, o sentirse seguro por ser observado [30].

Figura 17: Comando Actualidad: "Nos vigilan". RTVE

Es en este punto donde cobra ventaja la aplicación que se realiza, ya que permite realizar vigilancia en cualquier tipo de localización, basándose, únicamente en un teléfono móvil, elemento que en la sociedad desarrollada actual, está considerado por los habitantes como indispensable. Y por supuesto, a un precio mucho menor, y sin necesidad de involucrar a terceras persona u empresas en la seguridad privada.

2.3.3. ¿Es innovadora esta aplicación?

En cuanto al concepto de sistema que detecta movimiento de las imágenes que toma la cámara y las envía en diferentes formatos a otros dispositivos, la respuesta es NO. Debido a que existen más aplicaciones con estas características, entre las que destacan Digia ImageSpy y Spymanager.

La primera referencia encontrada de Digia ImageSpy data de 2002, desarrollada para Symbian, con un precio de 14.95 $, con un requisito de 170 Kb de memoria [12]. En sus características, se dice que se trata de una aplicación con un algoritmo de detección de movimiento (no indicado). Se activa por detección de movimiento o por tiempo, dispone de varios tipos de avisos, alarma sonora, SMS, MMS o llamada, almacenamiento de imágenes enviadas y desactivación por código. En esencia cumple las características de comunicación y detección de la aplicación que se desarrolla en este proyecto, pero no permite el manejo remoto de la aplicación por parte de otro dispositivo así como no existe ningún tipo de comunicación con un servidor Web en el que almacenar de forma permanente las imágenes y poder visualizarlas cuando así se quiera. Además, en los foros visitados, se destaca la imposibilidad de instalación y el número reducido de terminales en el que funciona (según especificaciones solo en Nokia 7650/3650). No se ha encontrado ninguna versión de prueba estable que funcione en otro terminal, por lo que no se ha podido realizar la comprobación de si las especificaciones mostradas se llevan a cabo. Además, dentro de la empresa de desarrollo Digia (autora de la aplicación), no se encuentra ninguna referencia a ella [11].

Figura 18: Digia ImageSpy

La segunda aplicación, Spymanager [29], desarrollada en su primera versión en 2006. Se trata de una aplicación que tiene una doble funcionalidad, en modo servidor recibe las imágenes de una WebCam conectada a un ordenador, mediante un

servidor propio (que también hay que descargar e instalar); esta funcionalidad no ha sido probada. Mientras que en el modo cliente, se permite la toma de imágenes por intervalo de tiempo o por detección de movimiento, y la imagen se envía a través de bluetooth a otro dispositivo o en un correo electrónico. Ambos envíos hay que configurarlos previamente. Esta parte si ha sido probada, no se han detectado fallos en la detección de movimiento (tampoco se indica el algoritmo utilizado). En cuanto a la usabilidad, quizá se hace demasiado pesada la configuración hasta que se llega al cuerpo real de la aplicación (la detección), debiendo realizar demasiados pasos cada vez que se inicia la aplicación.

Figura 19: SpyManager

En comparación con la aplicación desarrollada en este proyecto, carece de la comunicación entre dispositivos mediante mensajes multimedia, y por supuesto tampoco tiene el control remoto de dispositivo. El apartado visual proporcionado por LWUIT es más atractivo, y no necesita una configuración ni realizar tantos pasos para cada uso. En cuanto a la utilización del servidor, en esta aplicación se trata de un servidor Web al alcance de cualquier conexión a Internet, mientras que Spymanager utiliza uno propio.

Por tanto, se puede concluir que el concepto no es innovador, pero supone un sistema que aúna las características de las aplicaciones encontradas, añadiendo el control remoto entre dispositivos, y abriendo el servidor a Internet.

3. Análisis de la aplicación

Este apartado muestra la funcionalidad de la aplicación sin entrar en detalle de como se ha programado.

3.1. Esquema

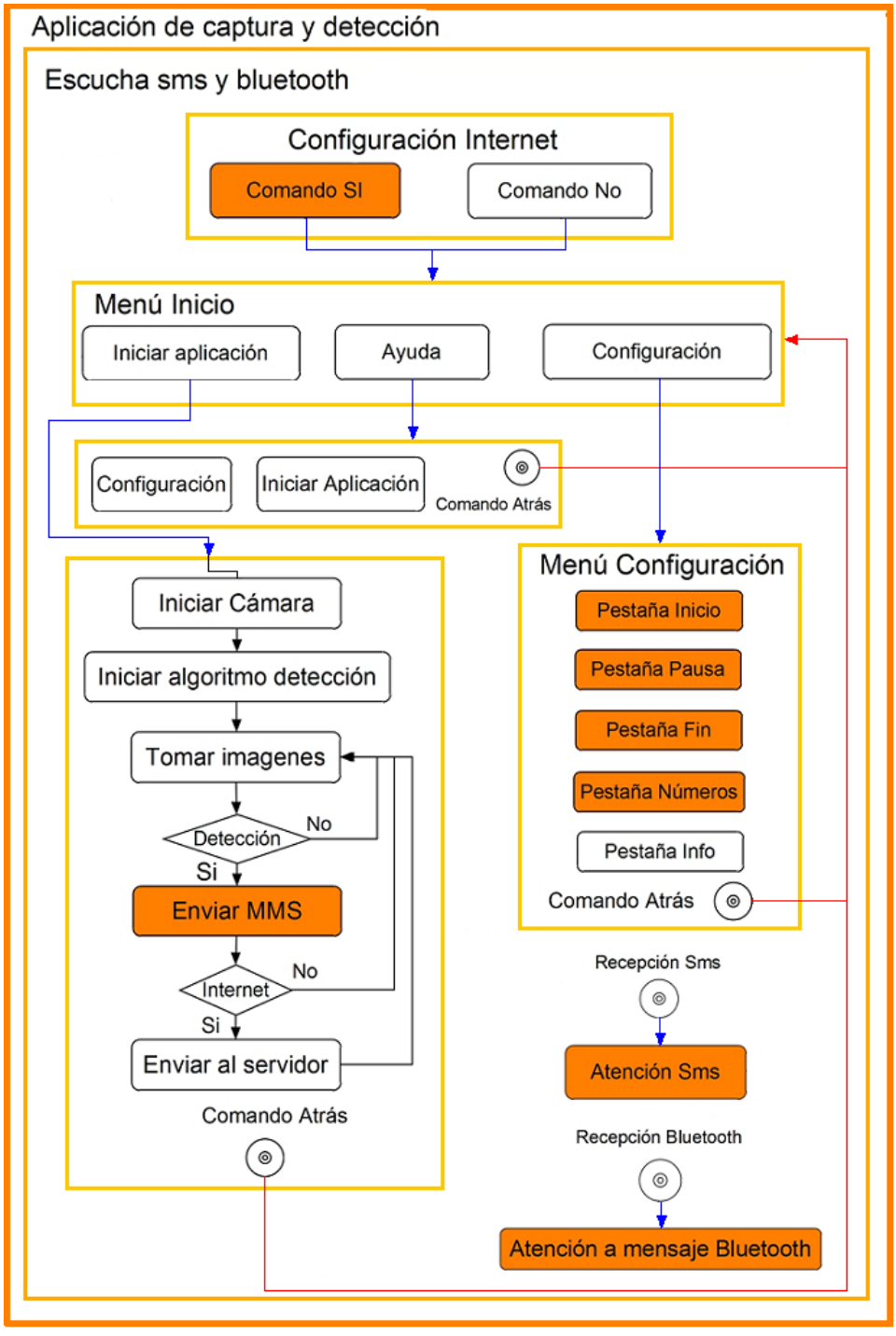

Figura 20: Esquema aplicación captura

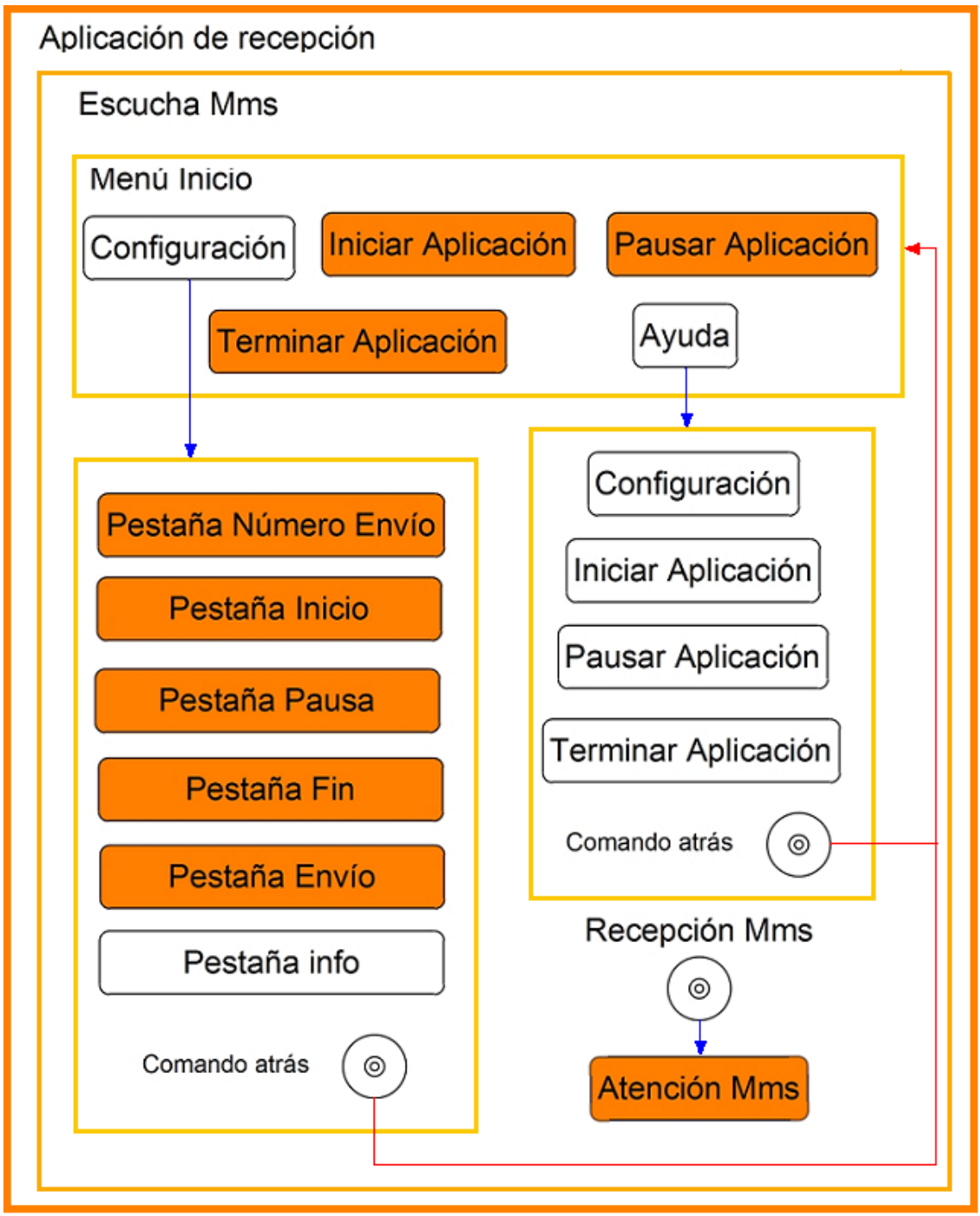

Figura 21: Esquema aplicación recepción

3.2. Emisor

3.2.1. Arranque

Nada mas lanzar la aplicación de captura, se procederá a crear los mecanismos de gestión de comunicación mediante mensajes de texto y bluetooth. La aplicación, si está abierta, tendrá un hilo de escucha a los mensajes de texto que lleguen al teléfono móvil, analizándolos y actuando en función del contenido de los mismos [ver Tabla 2]. Para el bluetooth, en este caso, el terminal funcionará en modo servidor. Esto es, que tendrá creado un servicio en su interior, que se ejecutará cuando llegue el mensaje de conexión bluetooth adecuado. Dicho servicio consiste en comparar la palabra que viene en el mensaje, con la palabra fin de la aplicación y apagar la aplicación en caso de que coincidan.

3.2.2. Menú Inicio

Una vez iniciadas las comunicaciones aparecerá una pantalla de configuración de red, en ella se podrá seleccionar si se quiere o no activar la comunicación con el servidor a través de Internet. Para intentar realizar la conexión es necesario introducir el número de teléfono del terminal y pulsar el comando “Si”.

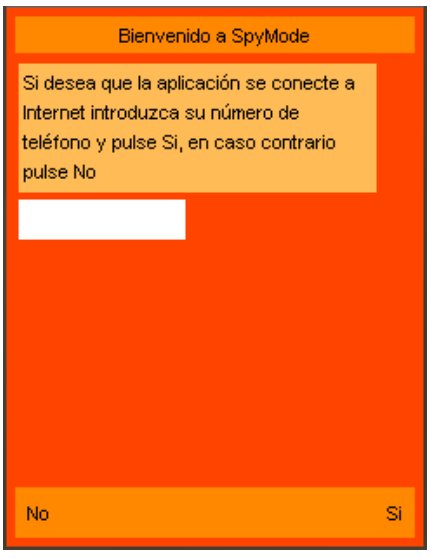

Figura 22: Menú Inicio Captura

El número será válido si consta de 9 dígitos, en caso contrario aparecerá un mensaje de error indicando que debe introducir un número con esa longitud [ver Figura 20]. Se establecerá la conexión con el servidor enviando el número introducido en el cuadro de texto y será el servidor el que compruebe la validez del mismo. Mediante códigos de error de la conexión se comunica al terminal los motivos por los que no se ha podido establecer la conexión con el servidor.

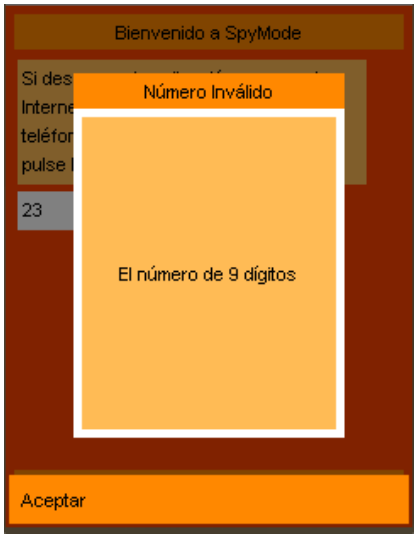

Figura 23: Menú Inicio Captura, error número

Los errores pueden ser:

- La conexión se realiza de manera adecuada pero el número no se encuentra en la base de datos, realmente no es un fallo de conexión propiamente dicho, pero si es un fallo en la autentificación del usuario, y por tanto, este funcionamiento no puede ser activado.

- La conexión se establece pero el servidor esta fuera de servicio, es decir, existe capacidad de conexión, pero por un fallo de servidor no se realiza la conexión.

- La conexión no se puede realizar, el terminal no es capaz de alcanzar al servidor, o no tiene las capacidades o características necesarias para poder realizar una conexión de este tipo.

Para cualquiera de los tres casos de error, aparecerá un cuadro de texto en la parte inferior de la pantalla indicando el motivo del error.

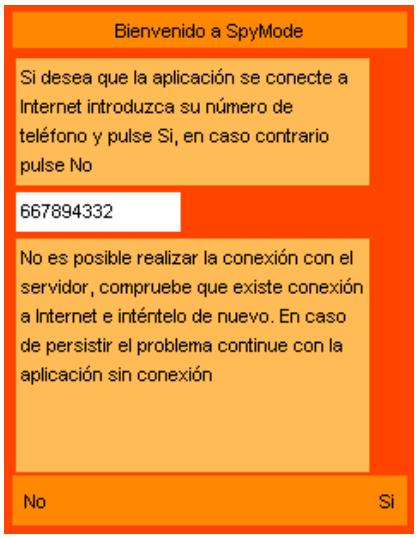

Figura 24: Menú Inicio Captura, error de conexión

Si todo el proceso es correcto, se accederá a la siguiente pantalla, Menú Captura. En este momento la aplicación conocerá que es posible la conexión con el servidor, y por tanto, cuando proceda, le enviara la información necesaria.

Si no se desea conectar con el servidor, en la pantalla inicial se deberá pulsar el comando "No", se accede a la pantalla de Menú Captura, pero en la aplicación esta desactivada la función de comunicación con Internet.

3.2.3. Menú Captura

Esta pantalla permite la navegación por las diferentes opciones que tiene la aplicación de captura, mediante tres botones:

- Iniciar Aplicación: Se inicia el mecanismo de captura de imágenes y detección de movimiento

- Menú Configuración: Se accede a la pantalla de configuración de la aplicación de captura.

- Ayuda: Se accede a la pantalla de ayuda global de la aplicación de captura.

Mediante el comando "Salir" se cerrará la aplicación y se liberarán los recursos temporales que se hayan utilizando, quedando en memoria del teléfono los datos que así se hayan introducido (números de teléfono y palabras clave).

Figura 25: Menú Captura

3.2.4. Iniciar Aplicación

Se trata del acceso al corazón de la aplicación, el resto de opciones se basan en establecer una mínima configuración interactuando con el usuario, pero es esta parte, la que realiza la verdadera funcionalidad de la aplicación, es la que realiza la captura y detección de movimiento.

Por tanto, al pulsar el botón, en un primer momento, se esperará un tiempo prudencial para que se pueda colocar el dispositivo móvil en el lugar final, en la zona a "vigilar". Esta espera es sumamente necesaria, ya que, de no producirse, en el transcurso desde que se inicia la aplicación hasta que se coloca el dispositivo en su ubicación adecuada se producirían numerosas detecciones, que supondría el envío masivo de mensajes multimedia, y por tanto un gasto innecesario de dinero.

Por tanto, la ejecución se sucede cronológicamente de la siguiente manera: Pulsar el botón de Inicio → Acceso a la cámara → espera → inicio detección.

Una vez de da inicio a la detección, también se espera un cierto periodo temporal, mucho menor que el anterior, con el fin de que el algoritmo este preparado para iniciar a detectar, y se tomen las imágenes adecuadas. Podría ocurrir, de no esperar este tiempo, que la primera imagen tomada, sea en el acceso de la cámara o cuando aun no se ha estabilizado el sistema, y por tanto, la primera detección sea una falsa alarma (y una falsa alarma tiene un coste proporcional al número de mensajes multimedia que ocupe la imagen). Además, este tiempo, supone un margen de seguridad respecto al primer tiempo de espera.

Una vez ya está colocado el dispositivo, y se han producido todas las esperas necesarias, la cámara comenzara a tomar imágenes de manera continua y a dictaminar si hay o no movimiento con el algoritmo basado en la distancia de Manhattan [7].

Esta distancia se basa en obtener la diferencia global de color entre dos puntos. Viene determinada por la siguiente fórmula:

$$d_{MAN} = |(R_1 - R_2) + (V_1 - V_2) + (A_1 - A_2)|$$

Donde R es el componente de rojo, V el componente de verde y A el componente de azul

Con ella obtenemos el valor absoluto de la diferencia de las componentes RGB de los dos puntos.

Se han determinado 9 puntos de posible detección, lo ideal sería utilizar un algoritmo de detección global, pero un teléfono móvil es un dispositivo de recursos bastante limitados, y una carga computacional tan elevada, conduciría a un bloqueo del terminal y al agotamiento casi instantáneo de la memoria del teléfono, hecho de por si totalmente indeseable.

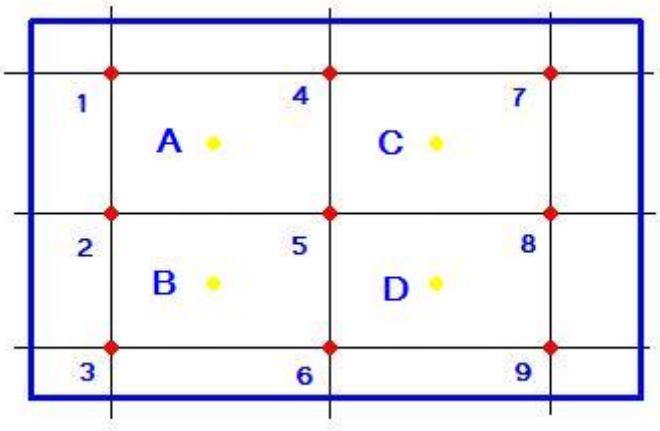

Figura 26: Puntos detección movimiento

Con estos nueve puntos, cualquier objeto o persona que aparezca en el campo de visión de la cámara, será detectado, exceptuando que sea tan sumamente pequeño para que pase entre dos de los puntos sin afectar a ninguno de ellos. Quizá para este caso, sería recomendable utilizar los puntos A, B, C o D. Pero supondría cargar más computacionalmente el terminal.

En el momento en que se produzca una detección, se procederá siempre al envío mediante MMS de la imagen en la que se ha producido el cambio, se comparan dos imágenes y se entiende como imagen de cambio, la segunda de ellas. El envío se realizará a tantos teléfonos como así se haya indicado en el menú de configuración en el apartado de números. Para evitar que se produzca el envío masivo de imágenes cuando se produce una detección, se realiza una espera en el hilo de captura después de iniciar el hilo de envío. Esta espera es necesaria, ya que, lo más lógico, es que si hay un cambio en la imagen, en una zona, el propio movimiento del elemento extraño se produzca por todo el campo de visión del móvil, y es un gasto innecesario el envío de 9 posibles

mensajes, uno por punto de detección. Tras esta espera, se reiniciara el proceso de captura.

En caso de que exista algún problema con el envío del mensaje, no se bloqueará la aplicación de captura ni de detección, ya que ambas deben estar en continuo funcionamiento. El único problema reside en que si no se ha introducido ningún número como destinatario, se lanzará un mensaje de error y no se seguirá ejecutando la aplicación, hasta que el usuario confirme que quiere proseguir con ella [ver Figura 24]. La justificación de esta interrupción en la aplicación es, que, el fin de la detección es informar al otro terminal de que ha pasado algún suceso extraño en la zona que se está vigilando, sino hay destinatario, no hay comunicación y por tanto la aplicación no cumple su función y no es necesario que se siga ejecutando.

Figura 27: Iniciar Aplicación, fallo número envío inexistente

Además del envío mediante MMS, si se ha seleccionado la opción de comunicación con el servidor y esta se ha llevado a cabo de manera satisfactoria, también, dicha imagen será enviada al servidor junto con el número de identificación del terminal, número propio de teléfono, para así guardarlo en la base de datos. En caso de que se produzca algún error en el envío la aplicación no se bloqueara, y seguirá con su ejecución, solo que la información no quedara registrada en el servidor.

Se dispone de un comando "Atrás" para evitar posibles accesos inesperados a la captura y detección que supongan una falsa alarma, con el cual se volverá al menú principal de la aplicación. Así una entrada inoportuna no generara un gasto económico si se produce una rápida reacción del usuario.

Figura 28: Iniciar Aplicación, vista cámara

3.2.5. Menú Configuración

Dentro de esta pantalla se pueden introducir o modificar las palabras clave para el acceso remoto de la aplicación, así como los números de teléfono a los cuales se enviará el mensaje multimedia en caso de que se produzca una detección.

Tanto las palabras (inicio, pausa, fin), como los números (tres, dos o uno), se guardarán en la memoria interna del teléfono a fin de que, salvo que en algún momento se quiera cambiar, sólo será necesario realizar la configuración una primera vez, quedando permanentes dichos datos para el resto de ejecuciones.

En la pantalla se pueden observar dos niveles de navegación, gestionados con pestañas. El primero de ellos permitirá elegir entre la configuración propiamente dicha o una pantalla de ayuda con la explicación de cómo y para qué rellenar los

diferentes campos. El segundo nivel, es la configuración propiamente dicha, se podrá navegar entre las pestañas de Inicio, Pausa, Fin y números.

Figura 29: Menú Configuración Captura, niveles de representación

Las tres primeras pestañas del segundo nivel, correspondientes a las palabras claves, ofrecen el mismo diseño y la misma funcionalidad.

En un cuadro de texto se introduce la palabra deseada, y mediante un botón se confirma que se desea guardar dicha palabra en memoria. No existe ningún tipo de limitación en cuando a longitud o formato de la palabra, si bien se recomienda que sea un vocablo sencillo y fácil de recordar, ya que deberán ser iguales a las que se introducen en la aplicación del receptor, si se quiere que el control remoto sea satisfactorio.

Tampoco existe ningún tipo de obligación en cuanto a la diferencia entre las tres palabras, es decir, que la palabra de inicio, pausa y fin, puede ser la misma.

Una vez se pulsa el botón de guardar, si no se produce ningún error, aparecerá un mensaje de confirmación.

Figura 30: Menú Configuración Captura, palabra guardada

En caso de equivocación, y se quiera guardar una palabra en blanco, aparecerá un mensaje de error indicando que este caso no es válido.

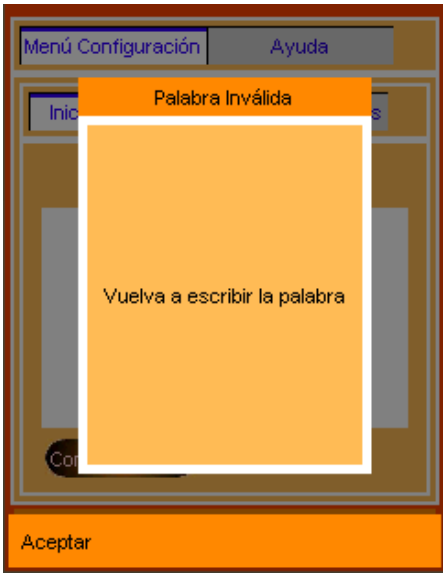

Figura 31: Menú Configuración Captura, palabra inválida

En cuanto al apartado de números, visualmente aparecen tres cuadros de texto, uno para cada posible número, al cual se enviará el mensaje multimedia con la imagen, en caso de que se produzca una detección. Hay que destacar, que si se introduce un número, siempre se enviará el mensaje multimedia a dicho teléfono, con el coste asociado que supone, por lo que no se debe utilizar de manera indiscriminada.

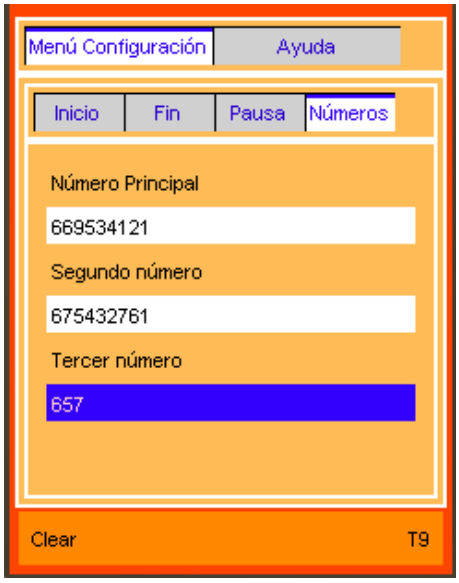

Figura 32: Menú Configuración Captura, pestaña números envío

Cuando se accede a esta pestaña, en la zona derecha de la barra de menú aparece un nuevo comando "Guardar números", que desaparecerá cuando se salga de esta pestaña, y con el cual, se intentaran guardar los números introducidos en los cuadros de texto, siempre y cuando cumplan una serie de requisitos.

Dichos requisitos son:

- Los números deben constar de nueve dígitos.
- Los números deben ser diferentes entre sí
- Al menos, el primero de ellos, el número principal, debe existir.

En caso de que no se cumplan estos requisitos aparecerá un mensaje de error indicando el motivo por el cual no se ha producido el guardado de los números en la memoria.

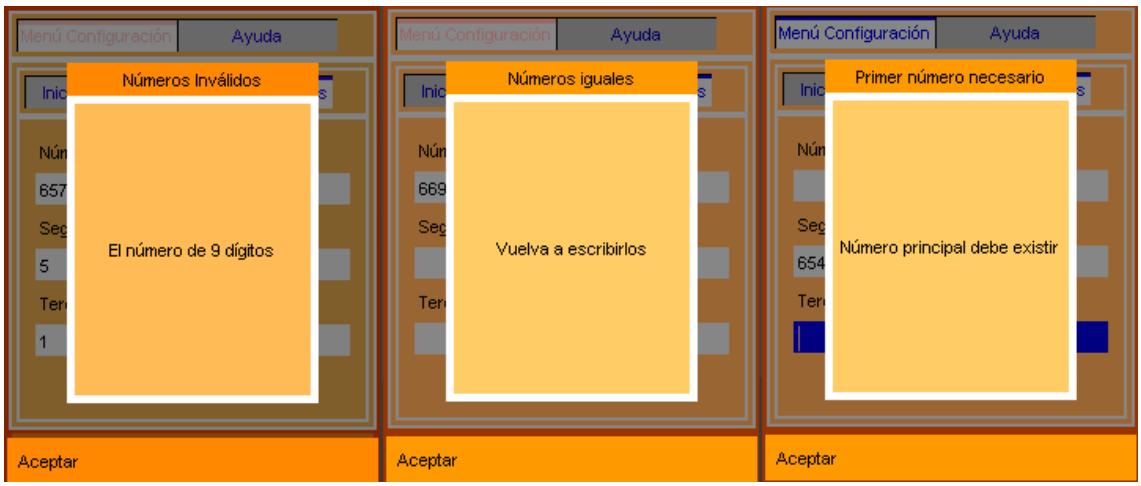

Figura 33: Menú Configuración Captura, pestaña números, errores

Si se cumplen los requisitos, y el proceso de almacenamiento se realiza de forma correcta, aparecerá un mensaje de confirmación.

Figura 34: Menú Configuración Captura, pestaña números, confirmación almacenamiento

Para el envío del mensaje multimedia, es necesario, que el numero empiece con el prefijo "+34", por lo que, para evitar que sea el usuario el que tenga que recordar este detalle, que incluso puede desconocer, la aplicación internamente añadirá el prefijo al número antes del envío.

Independientemente de la pestaña en la que se esté situado, siempre aparecerá un comando "Atrás", que al pulsarlo volverá al menú principal de la aplicación.

3.2.6. Ayuda

En esta pantalla se muestra la ayuda general de la aplicación de captura. El diseño consta de dos pestañas en la zona derecha de la pantalla, por las que navegar dependiendo de la funcionalidad que se quiera consultar. Y en la parte derecha aparece el contenido de la pestaña seleccionada. En el caso en que el texto supere el tamaño de la pantalla, se podrá acceder al resto del texto desplazándose verticalmente por la pantalla.

Mediante el comando "Atrás" se volverá al menú de la aplicación de captura.

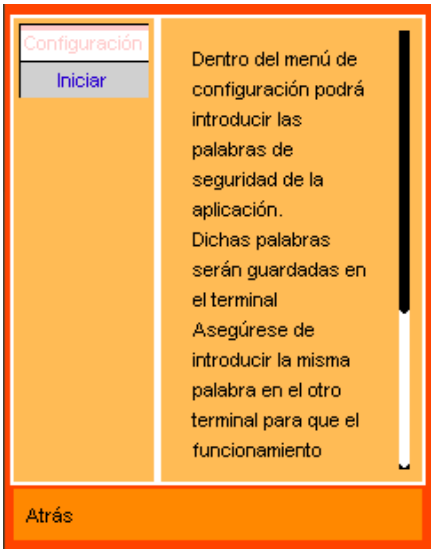

Figura 35: Ayuda Captura

3.2.7. Escucha SMS

Para el funcionamiento del control remoto vía SMS por parte del dispositivo en el que está instalada la aplicación de recepción, el emisor, debe estar siempre preparado para la recepción de un mensaje de texto, su tratamiento, y en caso de ser necesario, realizar la rutina de ejecución correspondiente.

Cuando llega un mensaje de texto, la aplicación lo abrirá y comprobará el texto que se incluye en él. Está establecido que cada mensaje dispone de un código especial inicial, para que la aplicación pueda descartar los mensajes que no sean dirigidos a ella.

Código	Nemónico rutina
qwegavdrtunm,nbvvgghhnnbaswertubz<aqwbcg	Guardar Inicio
ijybrnmkiuteszxz<<aqqwbnjklupñfjturnbvgy	Guardar Fin
fgtyhnmkopliutnmrtgfbvzcsdweqrahyutbhrto	Guardar Pausa
jurthaneritolanteqebcvrtiporntimncvbtrif	Guardar Todas
Palabra [2]+ iniciar	Iniciar Aplicación
Palabra (1) + pausar	Pausar Aplicación
Palabra (1) + terminar	Terminar Aplicación

Tabla 2: Correspondencia código-rutina

- Guardar Inicio:

  Se guarda en memoria la palabra de inicio que está incluida en el mensaje

- Guardar Fin:

  Se guarda en memoria la palabra de fin que está incluida en el mensaje

- Guardar Pausa:

  Se guarda en memoria la palabra de pausa que está incluida en el mensaje

- Guardar Todas:

  Se guardan en memoria las palabras de inicio, fin y pausa; que están incluidas en el mensaje.

- Iniciar Aplicación:

  En caso de que la aplicación de captura y detección no esté empezada, la inicia automáticamente.

- Pausar Aplicación:

  En caso de que la aplicación de captura y detección esté iniciada, la pausa, pudiéndose arrancar nuevamente en cualquier momento.

[2] *Palabra que se envía correspondiente a la palabra clave de la memoria donde se obtenga, bien sea, inicio, fin o pausa.*

- Terminar Aplicación:

  En caso de que la aplicación de captura y detección esté iniciada, la cierra. Sólo se podrá rearrancar nuevamente de forma manual, abriendo la aplicación de captura desde el principio

Cabe destacar, que todo mensaje que llegue a la aplicación, será abierto y procesado. Pero no todo mensaje que llegue tiene que pertenecer al funcionamiento de la aplicación. Todos los mensajes, pertenecientes o no, se almacenarán en el sistema que posee el teléfono móvil para ello, por lo que se recomienda, una vez cerrada la aplicación, comprobar el buzón de entrada de mensajes del dispositivo, por si ha llegado algún mensaje de texto que no está relacionado con la aplicación.

3.2.8. Escucha Bluetooth

Para el funcionamiento del control remoto vía bluetooth por parte del dispositivo en el que está instalada la aplicación de recepción, el emisor, debe estar siempre preparado para la recepción de un mensaje de texto vía bluetooth. Este caso solo se producirá para terminar la aplicación.

Se trata de una comunicación de corto alcance, sin coste adicional, por lo que supone una gran ventaja para desconectar la aplicación, poco antes de entrar en el campo de visión del teléfono.

Así, se puede maximizar el tiempo de funcionamiento de la detección y evitará el gasto asociado al envío de un mensaje de texto.

Una vez se ha terminado la aplicación, solamente se podrá volver a iniciar en el terminal donde está instalada, de la manera habitual, es decir, si se termina la aplicación de manera remota solo se podrá volver a iniciar de manera manual, por lo que si se utiliza esta opción, se ha de estar totalmente seguro de que no se volverá a utilizar hasta hacerlo manualmente. Funciona en modo servidor, dispone de un servicio

asociado que se encarga de la obtención de la palabra del mensaje y de la comparación con la clave, para en caso coincidente, apagar la aplicación.

3.3. Receptor

3.3.1. Arranque

Nada más lanzar la aplicación de recepción, se procederá a crear los mecanismos de gestión de comunicación mediante mensajes multimedia y bluetooth. La aplicación, si está abierta, tendrá un hilo de escucha a los mensajes multimedia que lleguen al teléfono móvil, analizándolos mostrando su contenido (referencia a como se actúa). Para el bluetooth, en este caso, el terminal funcionará en modo cliente, cuando le sea necesario utilizar un servicio de la aplicación que funciona como servidor, enviará el mensaje cuyo contenido contendrá los parámetros necesarios para que se realice el servicio.

3.3.2. Menú Inicio

Dentro de este menú, aparecerán una serie de botones que cubren la funcionalidad de la aplicación, mediante los cuales, se podrá acceder a la configuración, realizar la gestión remota de la aplicación de captura y detección, y tener disponible una ayuda global de la aplicación de recepción.

Existe un comando “salir”, con el cual se cierra la aplicación.

Figura 36: Menú Receptor

3.3.3. Menú Configuración

Dentro de esta pantalla, se podrá realizar la configuración necesaria para el funcionamiento de forma remota, además de permitir la configuración remota del otro terminal mediante el envío de mensajes de texto.

Sigue una estructura de pestañas por las cuales navegar en función del elemento que se desea configurar.

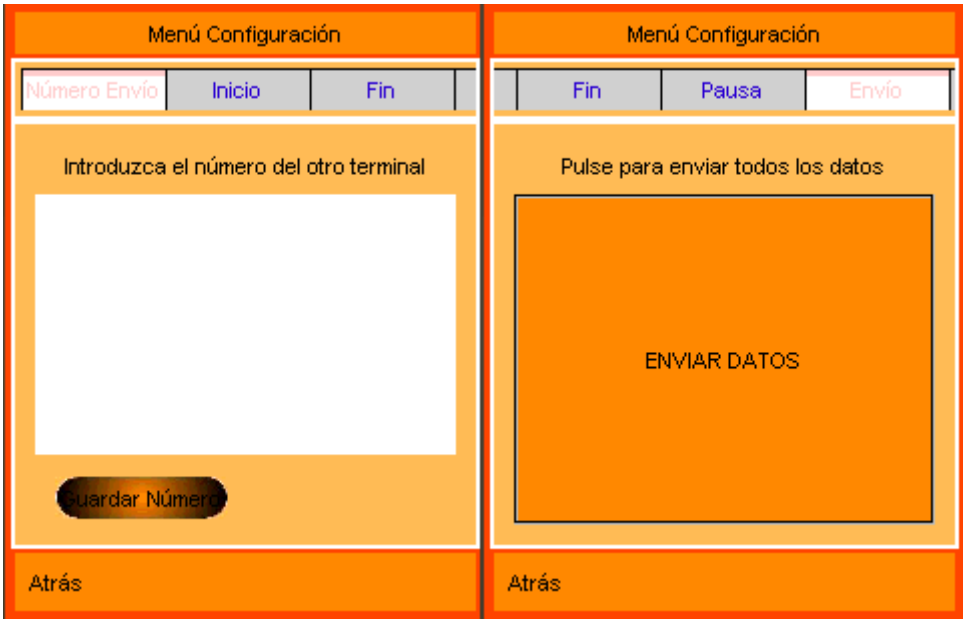

Figura 37: Menú Configuración Receptor

3.3.3.1. Número envío

Esta pestaña dispone de un cuadro de texto en el que se debe introducir el número de teléfono del terminal en que se va a utilizar la aplicación de captura y detección [ver figura 34].

Este número será utilizado para el envío de mensajes de texto, bien sean de configuración, o de funcionamiento remoto.

Se dispone de un botón "Guardar Numero", con el cual, se almacenará en memoria el número de teléfono si cumple el requisito de que tenga 9 dígitos, apareciendo un mensaje de error en caso contrario.

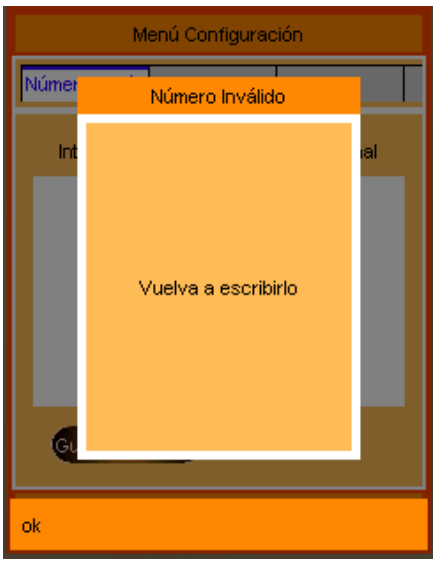

Figura 38: Menú Configuración Receptor, pestaña número, error

Si el número es válido, y no se produce ningún tipo de fallo en el almacenamiento del número aparecerá un mensaje de confirmación.

Figura 39: Menú Configuración Receptor, pestaña número, almacenamiento correcto

La realidad de este número, es decir, que sea el del terminal donde está instalada la aplicación de captura y detección, es imprescindible. Ya que de ser incorrecto, se estarán enviando mensajes de texto a un teléfono y a un usuario, que puede incluso desconocer la existencia de la aplicación de videovigilancia.

3.3.3.2. Pestaña inicio, pausa y fin

Las tres pestañas tienen la misma estructura: Un cuadro de texto para poder introducir la palabra clave y dos botones, uno para confirmar la palabra y otro para confirmar y enviar dicha palabra.

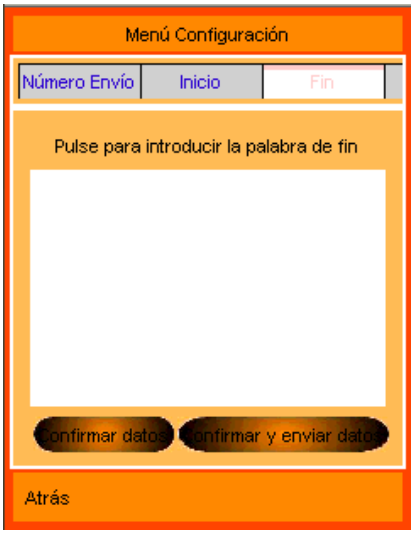

Figura 40: Menú Configuración Receptor, pestaña palabra fin

Cuando se pulsa el botón de confirmación, aparecerá en primera instancia un cuadro de texto, que en caso de que la palabra sea válida y se guarde de manera correcta en la memoria del teléfono, confirmará el proceso. Una palabra es válida si el cuadro de texto no se queda en blanco, en cuyo caso aparecerá un mensaje de error [ver figura 39]. No existe limitación de tamaño ni de caracteres utilizados, si bien se recomienda que sea una clave personal pero no difícil de recordar.

Figura 41: Menú Configuración Receptor, pestaña palabra fin, palabra correcta

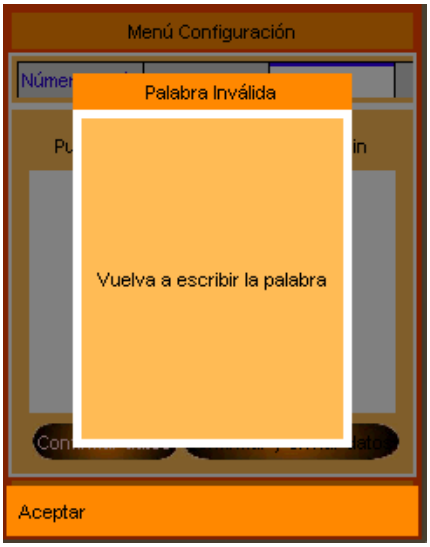

Figura 42: Menú Configuración Receptor, pestaña palabra fin, palabra incorrecta

Tampoco existe ningún tipo de limitación en lo que a la diferenciación de palabras se refiere, es decir, las tres palabras pueden ser exactamente

iguales, esto es posible, ya que se guarda cada palabra en una localización de memoria diferente, con un nombre que las caracteriza, y dicho nombre las diferencia.

Posterior al dialogo de confirmación, aparecerá en segunda instancia otro mensaje informativo, en cual se le pregunta al usuario si quiere o no enviar la palabra clave al otro terminal vía SMS. Si se produce el cambio de solo una de las palabras, es recomendable enviársela al otro terminal, salvo que se realice el cambio manual de la configuración de la aplicación de captura y detección.

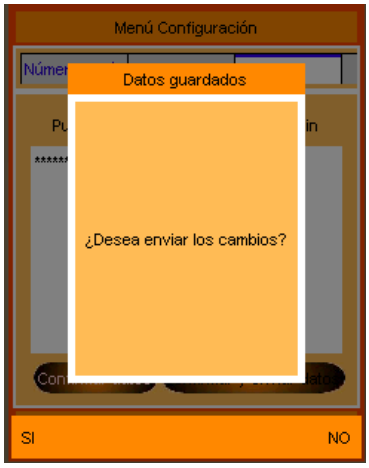

Figura 43: Figura 44: Menú Configuración Receptor, pestaña palabra fin, pregunta envío

En caso de aceptarse el envío, mediante un mensaje de texto con un código especial, se comunicará al otro terminal que debe cambiar su palabra clave por la que se indica en el mensaje. Si la comunicación es correcta aparecerá un mensaje de confirmación.

Figura 45: Menú Configuración Receptor, pestaña palabra fin, envío correcto

La repetición de los diálogos de confirmación de almacenamiento y de posible envío se produce en las tres pestañas, por lo que sí se está realizando un cambio global de todas las palabras de la aplicación, es recomendable, realizar todos los cambios de las palabras y acceder a la pestaña de envío (ver apartado 3.3.3.3). Así, evitaremos en envío de tres mensajes de texto por separado.

Con el botón de confirmar y enviar, se realiza el proceso completo de almacenamiento en memoria y envío por mensaje de texto al otro terminal, sin pedir ningún tipo de confirmación al usuario. La aplicación tomará la palabra del cuadro de texto, la guardara en el lugar adecuado en la memoria, y la enviará, junto con un código especial al otro terminal. Si todo este proceso es correcto, se indicará mediante un mensaje que ha sido satisfactorio [ver Figura 42].

3.3.3.3. Pestaña envío

Esta pestaña dispone de un único botón cuya funcionalidad es la de enviar todas las palabras que se encuentren en la memoria del teléfono, en un único mensaje de texto [ver Figura 34].

Enviará la palabras aunque no se hayan modificado, es decir, que mediante este botón, se accede a las tres zonas de memoria diferenciadas que almacenan por separado cada una de las palabras, las incluye en un mensaje de texto con un código especial, y las envía al otro terminal, el cual sustituirá las palabras que tiene por las que aparecen en el mensaje de texto recibido.

Esta funcionalidad es recomendable en dos casos:

- Si se está realizando por primera vez la configuración, y se desea configurar de manera remota la aplicación de captura y detección, entonces en vez de enviar tres mensajes de texto, uno por palabra que se introduce, se envían las tres en un único mensaje.

• Si se procede al cambio de dos o tres palabras claves en el mismo proceso de confirmación, y se desea realizar la configuración remota del otro terminal, cambiando las palabras antiguas por las nuevas.

En resumen, será conveniente, para enviar en un solo mensaje lo que de manera particular se enviaría en dos o tres mensajes de texto.

Al finalizar el proceso, si se realiza de manera satisfactoria, también se indicara con un dialogo [ver Figura 42].

3.3.3.4. Pestaña info

En esta pestaña aparecerá la información relativa al uso y características del resto de pestañas, se indicara en un cuadro de texto, que al ser mayor que la pantalla se desplazará verticalmente.

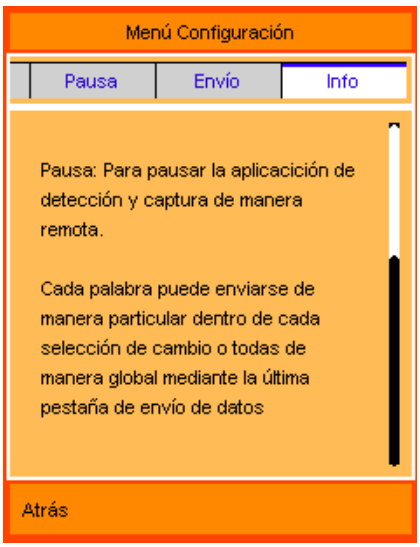

Figura 46: Menú Configuración Receptor, pestaña info

3.3.4. Iniciar aplicación

Con este botón, controlamos el inicio de la aplicación de captura y detección del otro terminal de manera remota.

Para ello, internamente, la aplicación tomará la palabra de inicio y enviará un SMS con un código especial asociado, al otro terminal, el cual será capaz de interpretar y actuar en consecuencia.

En caso de que no exista la palabra de inicio, o que se produzca algún problema en el acceso a la misma, aparecerá un mensaje de error, indicando, que no existe la palabra, o que se ha llegado a una situación inesperada.

Figura 47: Palabra envío inexistente

Siempre que se produzca un error, sea de la naturaleza que sea, no se enviará el mensaje te texto, ya que sería un gasto innecesario, puesto que no cumpliría con la funcionalidad esperada.

Para que la comunicación sea la correcta, es totalmente necesario, que coincidan, exactamente, las palabras de inicio de la aplicación de captura y de la aplicación de recepción. Si esto no ocurriera, se enviaría un mensaje de una aplicación a otra, pero no se obtendría ninguna funcionalidad, y podría llevar a error al usuario, creyendo que la aplicación está funcionando, y no es así.

En el caso de que no se produzca ningún error en la obtención de la palabra, esta exista, y produzca satisfactoriamente el envío, aparecerá un mensaje de confirmación de envío.

Figura 48: Envío correcto

Si todo es correcto, en este momento, la aplicación de captura y detección iniciará su funcionamiento. Este inicio, se produce solo en el caso de que dicha aplicación ya esté arrancada en el otro terminal, y se encuentre en un estado pausado. SI la aplicación no está arrancada, no se iniciará.

3.3.5. Pausar aplicación

Con este botón, controlamos la pausa de la aplicación de captura y detección del otro terminal de manera remota.

Para ello, internamente, la aplicación tomará la palabra de pausa y enviará un SMS con un código especial asociado, al otro terminal, el cual será capaz de interpretar y actuar en consecuencia.

En caso de que no exista la palabra de pausa, o que se produzca algún problema en el acceso a la misma, aparecerá un mensaje de error, indicando, que no existe la palabra, o que se ha llegado a una situación inesperada [ver Figura 44].

Siempre que se produzca un error, sea de la naturaleza que sea, no se enviará el mensaje te texto, ya que sería un gasto innecesario, puesto que no cumpliría con la funcionalidad esperada.

Para que la comunicación sea la correcta, es totalmente necesario, que coincidan, exactamente, las palabras de pausa de la aplicación de captura y de la aplicación de recepción. Si esto no ocurriera, se enviaría un mensaje de una aplicación a otra, pero no se obtendría ninguna funcionalidad, y podría llevar a error al usuario, creyendo que la aplicación está funcionando, y no es así.

En el caso de que no se produzca ningún error en la obtención de la palabra, esta exista, y produzca satisfactoriamente el envío, aparecerá un mensaje de confirmación de envío [ver Figura 45].

Si todo es correcto, en este momento, la aplicación de captura y detección parará su funcionamiento. Esta pausa, evidentemente, será solo efectiva si la aplicación esta arrancada y con el sistema de detección y captura en funcionamiento. Al pausar la aplicación, no salimos de ella, por lo que en cualquier momento, se podrá iniciar nuevamente de manera remota o cerrarla definitivamente, si así se quiere.

3.3.6. Terminar aplicación

Con este botón, controlamos la finalización de la aplicación de captura y detección del otro terminal de manera remota.

En vez de enviar directamente el mensaje, se puede decidir la forma de finalización. Estas maneras son mediante bluetooth o mediante un mensaje de texto, opciones, que aparecen en un dialogo de selección, una vez se pulse el botón.

Figura 49: Terminar aplicación, diálogo selección

3.3.6.1. Bluetooth

Si se selecciona la opción de finalización mediante bluetooth, la aplicación tomará la palabra de finalización y la enviará mediante este sistema a la aplicación de captura y detección.

El acceso a la palabra es igual al del envío por SMS, es una acción común, por lo que necesita los mismos requisitos, que exista, que sea accesible, y para su correcto funcionamiento, que sea igual en ambos terminales.

Para que esta funcionalidad sea satisfactoria, se debe estar dentro del campo de actuación de la comunicación bluetooth, y que ambos terminales tengan activa dicha comunicación. A la hora del envío, no es necesario realizar ningún tipo de vinculación clásica de dispositivos, puesto que están definidos internamente en las propiedades de la aplicación.

En el caso de que no se produzca ningún error en la obtención de la palabra, esta exista, y produzca satisfactoriamente la comunicación y el envío, en este momento, la aplicación de captura y detección se cerrará. En caso de no existir la palabra se indicara con un mensaje de error [ver Figura 44].

Esta finalización, evidentemente, será solo efectiva si la aplicación esta arrancada, siendo indiferente su estado, es decir, puede estar en modo pausa, o en modo activo, que se cerrará irremediablemente. Al cerrar la aplicación, salimos de ella, pudiendo solo restablecerla de forma manual.

3.3.6.2. SMS

Si se selecciona la opción de finalización mediante SMS, la aplicación tomará la palabra de finalización y la enviara mediante un mensaje de texto.

Para evitar el envío innecesario y su gasto asociado, si se produce algún error en el acceso a la palabra, o si esta no existe, nunca se enviará el mensaje, apareciendo un mensaje de error [ver Figura 44].

Si todo el proceso de validez de palabra, acceso, y envío, es satisfactorio, aparecerá un mensaje de confirmación de envío [ver Figura 45].

Para el correcto funcionamiento, como en el resto de los casos, es totalmente necesario que las palabras de fin sean iguales en ambos terminales.

Si todo es correcto, en este momento, la aplicación de captura y detección se cerrará. Esta finalización, evidentemente, será solo efectiva si la aplicación esta arrancada, siendo indiferente su estado, es decir, puede estar en modo pausa, o en modo activo, que se cerrará irremediablemente. Al cerrar la aplicación, salimos de ella, pudiendo solo restablecerla de forma manual.

3.3.7. Ayuda

Al pulsar este botón, aparecerá una nueva pantalla, en la cual se podrá encontrar el texto de ayuda global asociado a la aplicación.

La distribución será similar a la pantalla de ayuda de la aplicación de captura y recepción, con un grupo de pestañas navegables en la zona derecha, y la información asociada a cada una de ellas en la zona izquierda dentro de un cuadro de texto.

Dispone de un comando "Atrás", con el que volver a la pantalla del menú Inicio.

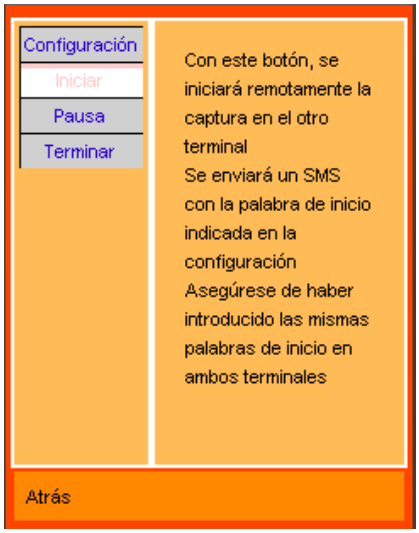

Figura 50: Ayuda Receptor

3.3.8. Escucha MMS

En cualquier momento, el otro terminal, donde esta ejecutándose la aplicación de captura y detección, puede enviar un mensaje multimedia con la imagen en la que se ha detectado movimiento. En el caso de que la aplicación receptora esté abierta, será capaz de recibir y procesar dicho mensaje. Lo abrirá, comprobara la cabecera, para dictaminar si es un mensaje con la imagen o si se trata de un mensaje multimedia ajeno a la aplicación. En caso de ser la imagen enviada por la aplicación de detección y captura, este en la zona que este de la aplicación de recepción, aparecerá un cuadro de dialogo con la imagen detectada acompañado de un sonido de aviso.

3.3.9. Bluetooth

Dentro del menú principal se crea el objeto necesario para la comunicación bluetooth (en este caso Bluetooth local), dicho objeto, en sí, no realiza más acción que dar las propiedades que luego se utilizarán para el envío. Por sí solo, no tiene ninguna función más, no realiza ningún tipo de escucha. Cuando sea necesario su uso, será el momento de buscar los dispositivos cercanos, adquiriendo los servicios que tienen en su interior, y enviando el mensaje adecuado al servicio de finalización que está alojado en la aplicación de captura y detección.

4. Descripción de la aplicación

Este apartado muestra como se ha programado específicamente la funcionalidad determinada en el apartado anterior, así como, la interfaz gráfica. En el anexo II, se encuentran los diagramas de flujos, para una mejor comprensión y seguimiento.

4.1. Emisor

4.1.1. Ayuda

En esta clase se crea el formulario sobre el que se incluyen los elementos visuales, la disposición en este caso consiste en un grupo de pestañas cuya navegación se sitúa en la parte izquierda y el contenido en la parte derecha.

```
public AyudaCaptura(Captura midlet_, final ConexionSms conexionSms)
```

En su constructor recibe dos parámetros:

- midlet_: Es un objeto de tipo Captura, es el midlet de la aplicación. En esta clase no se realiza ninguna acción sobre él, pero sí es necesario, puesto que debe estar presente en toda la aplicación y hay que transportarlo por todas las pantallas. En caso de no hacerlo así, al volver atrás perderíamos todo tipo de referencia a él, y por tanto no podría ser utilizado en otra pantalla donde sí es necesario realizar acciones sobre él.
- conexionSms: Es un objeto de tipo ConexionSms, tiene el control sobre la conexión de escucha de mensaje de textos, al igual que con el objeto anterior (midlet_), no se realiza ninguna acción sobre él, pero sí es necesario su transporte entre pantallas.

El cuerpo fundamental de la clase, sigue los siguientes pasos:

```
crearFormulario();
```

Se crea el formulario `(formulario = new Form();)` y se le añaden las características necesarias para esta pantalla, tales como la transición de

entrada y salida, en este caso con apariencia de cubo horizontal (`formulario.setTransitionInAnimator (Transition3D.createCube(400, true));`) y se evita la posibilidad de que la pantalla general, es decir, el formulario, pueda desplazarse verticalmente (`formulario.setScrollable(false);`).

```
crearPestañas();
```

Se crea el objeto TabbedPane que permite la existencia de variar pestañas dentro de un mismo formulario (o contenedor). En este caso, como se ha indicado, la navegación estará situada en la parte izquierda, `(pestañas = new TabbedPane(Component.LEFT);)`.

También, se le indica el tipo de transición al pasar de una pestaña a otra, en este caso será de apariencia de cubo vertical `(pestañas.setTransitionLeft(Transition3D.createVerticalCube(250, true));)`. Cabe destacar que el API de LWUIT solo ofrece la posibilidad de indicar la transición a izquierda o derecha. Al tratarse, en este caso de una transición arriba-abajo, la equivalencia es de izquierda-abajo, derecha-arriba.

Se evita que el propio contenedor de pestañas pueda desplazarse verticalmente del mismo modo que en el formulario, y por último se crean y añaden los componentes que se han de visualizar.

```
crearComponentesPestañas();
```

Dichos componentes serán contenedores, en el que el desplazamiento vertical sí estará permitido (no el horizontal), y tendrán incorporados un cuadro de texto (TextArea), con el texto asociado a cada uno de ellos.

```
configuracion = new Container();
configuracion.setScrollableY(true);
configuracion.setScrollableX(false);
areaConfiguracion= newTextArea(textoConfiguracion, 6, 20);
configuracion.addComponent(areaConfiguracion);
```

En el ejemplo se muestra el contendor para configuración, se procede de la misma manera para el otro componente, `iniciar;`

Una vez se han creado los componentes donde están todos los elementos que se quieren mostrar en la pestaña, hay que añadirlos al contenedor de pestañas, indicando con que nombre aparecerán, siguiendo con el ejemplo del contenedor configuración, se realiza con, `pestañas.addTab ("Configuración", configuracion);`. El orden en que aparecerán las pestañas, será el mismo en el que se incluyen.

```
darEstilo();
```

Mediante este método, se hacen llamadas a la clase Estilos, para que cada uno de los componentes tenga la apariencia que se desea. (poner referencia a clase estilos).

```
añadirComponentes();
```

Una vez están creados todos los objetos que se han de mostrar en el formulario es necesario añadirlos, se realiza con el método addComponent. Además de añadir el objeto pestañas, se incluye el comando atrás (método addCommand), y el escuchador de eventos de comandos (setCommandListener).

```
formulario.show();
```

Una vez esta todo creado, la única manera de que se visualice es indicándolo con este método.

Esta clase, además implementa la interfaz ActionListener, lo que le obliga a incluir el método `public void actionPerformed(ActionEvent evt)`. En cuanto se produzca algún tipo de evento, se atenderá en este método, siempre y cuando se hayan registrado los diferentes tipos de escuchadores, en este caso, como se ha indicado en el apartado de añadir componentes, solo se reacciona ante eventos de comandos, por

lo que la manera de registrar el escuchador de comandos será `(formulario.setCommandListener(this))`. Con this, indicamos que cuando se produzca el evento, el método encargado de su gestión es el actionPerformed.

La procedencia del evento se puede obtener del propio objeto ActionEvent que se genera, para así, dictaminar que acción hay que realizar. En este caso, cuando se pulse el comando atrás, se creará nuevamente el menú de captura. El comando atrás dispone de un identificador que se puede obtener, `evt.getCommand().getId()` para distinguirlo, y con una comparación, poder realizar la acción que está asociada a él.

4.1.2. Captura

Es el Midlet de la aplicación, y como tal, es la primera clase que será llamada al arrancar la aplicación.

```
public Captura(){
```

En su constructor únicamente se creará el objeto conexionSms, mediante el cual, se realizará la atención a los mensajes de texto que lleguen al terminal móvil mientras esté activa la aplicación (poner referencia a conexionSms).

```
}
```

Como todo midlet, ha de implementar los tres métodos del ciclo de vida:

```
public void startApp() {
```

Es el método mediante el cual se arranca el hilo de ejecución de la aplicación.

Para la utilización del apartado visual que proporciona LWUIT, es totalmente necesario e imprescindible realizar la llamada a `Display.init(this);`. Con esto, se arrancará el motor visual de LWUIT asociado al midlet, y así, se podrán utilizar todas sus características.

Posteriormente se le dotará a la aplicación de la escucha de la comunicación bluetooth `servidor = new BluetoothRemoto(this);`.

Y finalmente, se mostrará el menú principal de la aplicación `new MenuInicio(this, conexionSms);`

```
        }

        public void pauseApp() {
```

Es el método utilizado para pausar la aplicación cuando así se indique. Para esta aplicación, pausar significa, parar el mecanismo de captura de imágenes y detección de movimiento. Para ello, cuando se llama al método de pausa, lo primero que se hará será comprobar que esta activa la captura y detección, y en caso afirmativo, se realizaran los pasos necesarios para pararla, indicando al midlet debe parar la aplicación, y presentando en pantalla el menú correspondiente.

Para conocer si está o no activa la captura y detección, se utiliza el objeto pantallaVideo. Se trata del objeto encargado de dichas acciones, a lo largo de la aplicación, cuando se inicia, se indicará al midlet mediante el método `setPantallaVideo(PantallaVideo p)`. Con él se tendrá acceso a los elementos que controlan el desarrollo de su ejecución, para así, poder pararlos. En este caso, con `pantallaVideo.iniciar = false; pantallaVideo.contendorPlayer.stop();`, se conseguirá parar.

Para indicar que no está activa, es necesario actuar sobre el objeto pantallaVideo del midlet `setPantallaVideo(null);`.

Y finalmente, el menú que se mostrará, será el menú de inicio, creado cuando se termine el proceso de parada.

```
        }
```

```
public void destroyApp(boolean incondicional) {
```

Este es el método utilizado para indicar que pasos hay que realizar cuando se cierre la aplicación, en este caso, el único requisito es que se abandone, así que no es necesario implementar ninguna funcionalidad en él.

```
}
```

Dentro del midlet, también existe el método que será llamado cuando se reciba el mensaje de texto de inicio de aplicación, **public void** iniciar (). Este método comprobará si ya está activa la funcionalidad, a través del objeto pantallaVideo, y en el caso de que no esté iniciada, creara dicho objeto.

Como particularidad, cabe destacar el hecho de que la conexionSms no se cree dentro del hilo de ejecución principal, esto es debido a que el hilo de atención a mensajes es independiente al hilo principal de la aplicación y debe estar siempre activo y circular por la aplicación sin tener en cuenta el estado del midlet.

4.1.3. Detección

Esta clase es la encargada de dictaminar si hay o no diferencia de color entre las imágenes, deduciendo así, que existe algún tipo de movimiento.

```
public Deteccion(int [] rgb, int [] rgb2)
```

Recibe como parámetros dos arrays de enteros, que son los componentes de color de las imágenes.

Dentro del método **hayDeteccion()**, se encuentran las llamadas a los diferentes métodos que controlan los diferentes puntos de detección.

Existen nueve posibles localizaciones:

Píxel	Métodos
1	`deteccionArribaIzquierda()`
2	`deteccionIzquierda()`
3	`deteccionAbajoIzquierda()`
4	`deteccionArribaCentral()`
5	`deteccionCentral()`
6	`deteccionAbajoCentral()`
7	`deteccionArribaDerecha()`
8	`deteccionDerecha()`
9	`deteccionAbajoDerecha()`

Tabla 3: Localizaciones de detección

En todos los puntos, la manera de actuar es la misma. Se obtiene el color (2) de los puntos de estudio (1). Se trata de un String ya que posteriormente la información se puede dividir sin problemas en los tres componentes de color RGB (rojo, azul y verde) (3).

Una vez se separan los valores de color, se realiza una conversión a decimal del valor en binario que se obtiene (4), para así, poder ejecutar el algoritmo de comprobación.

A través del cálculo de la distancia de Manhattan (5), podemos comprobar si supera el umbral que se ha establecido, y así determinar si hay o no detección (6).

```
(1)   int pixel1 = rgbData[19199];
      int pixel2 = rgbData2[19199];

(2)   String color1 = Integer.toBinaryString(pixel1);
      String color2 = Integer.toBinaryString(pixel2);

(3)   String rojobin1 = color1.substring(8, 16);
      String verdebin1 = color1.substring(16, 24);
      String azulbin1 = color1.substring(24, 32);
      String rojobin2 = color2.substring(8, 16);
      String verdebin2 = color2.substring(16, 24);
      String azulbin2 = color2.substring(24, 32);
```

```
(4)   int rojdec1 = bintoDec(rojobin1);
      int verdec1 = bintoDec(verdebin1);
      int azdec1 = bintoDec(azulbin1);
      int rojdec2 = bintoDec(rojobin2);
      int verdec2 = bintoDec(verdebin2);
      int azdec2 = bintoDec(azulbin2);

(5)   distancia=Math.abs((rojdec1-rojdec2)+(verdec1-
      verdec2)+(azdec1-azdec2));

(6)   if( distancia>=umbral ){
            return true;
      }
            return false;
```

Una vez se han comprobado todos los puntos, se da el resultado final de la detección, en cuanto se haya producido en un punto, será afirmativa.

```
boolean comparacion = arribaizq || arribacent || arribader
|| centroizq || centro || centroder || abajoizq || abajocent ||
abajoder;

if( comparacion ){
      return true;
}
      return false;
```

4.1.4. Menú Captura

En esta clase se crea el formulario sobre el que se incluirán los elementos visuales, en este caso una serie de botones, mediante los cuales, acceder al resto de funcionalidades de la aplicación: Inicio, Configuración y Ayuda

```
public MenuCaptura(Captura midlet_, final ConexionSms conexionSms)
```

El constructor recibe los mismos parámetros que la clase de Ayuda. A la conexión SMS la trata de manera idéntica, mientras que del midlet, además de realizar su transporte al resto de la aplicación, utiliza la funcionalidad que proporciona el método destroyApp cuando se pulsa el comando salir.

El cuerpo fundamental de la clase, sigue los siguientes pasos:

```
crearFormulario();
```

Crea el formulario con la particularidad de que establece que el layout se divide en cinco filas y una única columna `(formulario.setLayout(new GridLayout(5, 1));)`

```
crearBotones();
```

Método que crea los botones que se incluirán en la pantalla y que servirán para navegar por las distintas funcionalidades de la aplicación.

Para la creación del botón se utiliza el constructor cuyo único parámetro es el texto que aparecerá en el.

Aunque la clase global implementa la clase ActionListener y se podría utilizar para que escuchara los eventos de cada botón, es preferible que cada botón posea su escuchador por separado para evitar conflictos a la hora de dictaminar que botón ha producido el evento, y para evitar una sentencia compleja a la hora de separar unos botones de otros.

Utilizando como ejemplo el botón de inicio, la creación y el registro del escuchador se implementarían de la siguiente manera:

```
inicio = new Button("Iniciar Aplicación");
inicio.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
      new PantallaVideo(midlet, conexionSms);}});
```

En este caso cuando se pulse el botón de inicio, se creará un nuevo objeto de tipo PantallaVideo, encargado de la captura de imágenes y detección de movimiento.

Para el caso del botón de Configuración, se creará un nuevo objeto de tipo MenuConfiguracionCaptura, que permite la posibilidad de realizar la configuración de seguridad y envío de la aplicación.

El botón de ayuda, creara un objeto de tipo AyudaCaptura, en cual se pueden consultar los textos informativos sobre el funcionamiento de la aplicación.

```
darEstilo();

añadirComponentes();
```

Además de los botones, que son añadidos en el orden en que aparecen en la pantalla, se añade el comando salir, así como el escuchador de comandos por defecto.

```
formulario.show();
```

Esta clase también implementa a la interfaz ActionListener, y por tanto posee un método actionPerformed, que también es utilizado únicamente para realizar las acciones que son consecuencia de los eventos generados por los comandos.

En este caso, solo existe un comando “Salir”, cuya funcionalidad es cerrar la aplicación de manera completa y definitiva (hasta que se vuelva a abrir de manera manual). Esto se consigue, en la rutina de atención al comando salir con:

```
midlet.destroyApp(true);
midlet.notifyDestroyed();
```

4.1.5. Menú configuración captura

Esta clase es la encargada de mostrar y gestionar la configuración de la aplicación, tanto en aspectos de seguridad como de envío. Por seguridad se entienden las palabras claves o contraseñas, que se utilizan para permitir que se produzca el manejo de esta aplicación de manera remota desde otro terminal en el que esté en funcionamiento la aplicación de recepción. Y envío se refiere a los números de teléfono a los cuales se enviaran los mensajes multimedia con la imagen resultante de una detección.

```
public MenuConfiguracionCaptura(Captura midlet_, ConexionSms
                                conexionSms)
```

Recibe los mismos parámetros que la clase AyudaCaptura, y los trata de la misma forma, es decir, no actúa sobre ellos, simplemente los transporta de una clase a otra.

El cuerpo fundamental de la clase, sigue los siguientes pasos:

```
crearFormulario();
```

Método donde se crea el formulario, se impide el desplazamiento vertical y se establecen las transiciones de entrada y salida de tipo cubo.

```
crearAreasTexto();
```

Método que crea los cuadros de texto que se incluirán en las pestañas de inicio, fin y pausa. Las tres se crean de la misma forma y poseen la misma característica en cuanto a la visualización de las palabras, estarán ocultas a modo de contraseña (*). Tomando como ejemplo el cuadro texto de la pestaña de inicio:

```
palabraInicio = new TextField();
palabraInicio.setConstraint(TextArea.PASSWORD);
```

```
crearPestañas();
```

La disposición de las pestañas será en dos niveles, el primero de ellos permitirá navegar entre las pestañas de configuración y ayuda, y el segundo entre los diferentes elementos de la configuración: Pestañas de inicio, pausa, fin y números.

El contenedor de pestañas principal tendrá situado sus campos de selección en la parte superior (pestañas = new TabbedPane (Component.*TOP*);), no permitirá el desplazamiento ni horizontal ni vertical, es decir, no modificará su tamaño. Cuando se utilizan varias pestañas, se pueden establecer acciones que ocurren cuando se pasa de una a otra pestaña, esto se realiza mediante un TabListener, el cual puede ser un SelectionListener que tiene asociado un método seclectionChanged, que captura la posición (pestaña) donde se encuentra la selección y la anterior que estaba seleccionada. Esto es muy útil si se quiere realizar alguna modificación o alguna funcionalidad cuando se cambia de una pestaña a otra. Utilizando como ejemplo la pestaña principal, la cual cuando cambia de la pestaña configuración a la de ayuda o viceversa, elimina el comando guardar números y sitúa como seleccionada la primera pestaña de configuración (en este caso inicio):

```
pestañas.addTabsListener(new SelectionListener() {
public void selectionChanged(int oldSelected, int newSelected) {
      formulario.removeCommand(CMD_SAVE);
      menuPestañas.setSelectedIndex(0);
}
});
```

La pestaña de ayuda tendrá en su interior un área de texto no modificable (`setEditable(false)`;) en el cual aparecerán las instrucciones necesarias para solventar cualquier duda en la configuración de la aplicación.

El contendor de pestañas secundario, también tendrá situado su campo de selección en la parte superior, y en su selectionChanged, tendrá como particularidad, que cuando se esté situado en la pestaña con el índice 3 (números),

aparecerá el comando guardar números, y cuando este situado en cualquier otro lugar, lo borrará.

La pestaña de números estará compuesta por una sucesión de títulos asociados a cuadros de texto donde poder introducir un número de teléfono.

Los títulos serán tratados como Label y tendrán su alineamiento a la izquierda:

```
num1 = new Label("Número Principal");
num1.setAlignment(Label.LEFT);
```

Los cuadros de texto tendrán como particularidad que solo permiten que se escriba en ellos caracteres numéricos, `numero1.setInputModeOrder(new String[] { "123" });`.

Una vez están creados todos los componentes, se añaden al contendor en el orden que se quieren mostrar (título, cuadro de texto). Dicho contenedor será el que finalmente se añada a la pestaña.

Las tres pestañas restantes, inicio, pausa y fin; siguen la misma composición. Un Label (etiqueta) con el título, un cuadro de texto donde introducir la palabra clave, y un contenedor donde estará alojado el botón confirmar.

Dicho botón tiene registrado un ActionListener que actuará cuando es pulsado [ver Figura 53].

Para cumplir la funcionalidad esperada para el botón, lo primero que se hace es obtener la palabra del cuadro de texto `(palabraInicio.getText(); )` se comprueba su validez `if (palabra != null && palabra.length() > 0))`, y en función del resultado se actúa de diversas maneras.

- válida:

Se procede a guardar la palabra en memoria (RecordStore). Para ello, se abre el RecordStore `(inicio =`

`RecordStore.openRecordStore("Inicio",true, RecordStore.AUTHMODE_PRIVATE, true);;` este método, abrirá el RecordStore en caso de que lo encuentre, y si no lo creará.

Una vez abierto, hay que comprobar si está o no lleno `if (inicio.getNumRecords() > 0).` Si lo está, habrá que cerrarlo (`closeRecordStore`), borrarlo `(RecordStore.deleteRecordStore("Inicio");)`, y crear uno nuevo con las mismas características pero vacío `(inicio = RecordStore.openRecordStore("Inicio", true,RecordStore.AUTHMODE_PRIVATE, true);).` Esto se realiza así, ya que es más sencillo atacar a la estructura completa RecordStore, que a su contenido, y puesto que hay que eliminar todo el contenido e incluir el nuevo, es parecido a eliminar y crear uno nuevo.

En caso de que esté vacío (y en el caso de ya haber realizado el proceso de borrado), se procederá a codificar la palabra (tipo String) a byte [] puesto que es el tipo de datos que acepta el RecordStore `(byte[] pInicio = p1.getBytes(); )` y se añadirá al mismo `inicio.addRecord(pInicio, 0, pInicio.length);.` Finalmente, se cierra el RecordStore.

Es importante que cada vez que se utilice un objeto RecordStore, se sea altamente escrupuloso con su estado, es decir, cuando está abierto y cuando no, ya que salir de la pantalla donde se crea el RecordStore no implica cerrarlo. Por lo que se recomienda, y es lo que se ha realizado, que al utilizar un RecordStore, se abra, se realicen las acciones necesarias sobre él, y se cierre. Así se evitan numerosos problemas, tanto de posibles duplicados como de accesos erróneos.

Si este proceso se produce con éxito, es decir, no salta ninguna excepción, aparecerá un dialogo de confirmación.

En el caso de que se produzca alguna excepción, aparecerá un dialogo por defecto con el tipo de fallo `Dialog.show("RecordStoreException", e.getMessage(), "Aceptar",null);.`

Para el caso exitoso, con un dialogo se trabaja igual que con un formulario, por tanto se debe crear, `Dialog confirmacion = new Dialog();`, añadirle el layout confirmacion.setLayout(new BorderLayout());, crear sus componentes y añadírselos (addComponent, addCommand), establecer su transición de entrada, en este caso de tipo vuelo, y finalmente mostrarlo (show();). También se puede establecer el tipo de dialogo, en este caso de confirmación, `confirmacion.setDialogType(Dialog.TYPE_CONFIRMATION);`. Cada tipo de dialogo, lleva asociado un tono musical característico.

- no válida:

En este caso, aparecerá otro dialogo, con las mismas características que el de confirmación, exceptuando el tipo, que en este caso será `Dialog.TYPE_ERROR.`

```
darEstilo();

añadirComponentes();
```

Dentro de este método se añaden los componentes, tal y como se desea que se visualicen, puesto que se quiere una que dentro de uno de los elementos de la pestaña principal exista otro contenedor de pestañas hay que indicarlo. Por tanto, el contendor de pestañas secundario contendrá los contendores de inicio, pausa, fin y números en este orden. Mientras que el contenedor principal, el primer elemento será el contenedor de pestañas secundario y el segundo será el contenedor de ayuda.

Al formulario hay que añadirle el contendor de pestañar principal, así como el comando “Atrás” y el escuchador de comandos por defecto.

```
ponerNumeroIniciales();
```

Con este método, se incluyen en la pantalla, en caso de existir, los números de teléfono que estén guardados en la memoria permanente del teléfono. Para ello, lo primero que hay que realizar es la comprobación de si existen o no en memoria,

y una vez se tienen, realizar un algoritmo para pintarlos. El algoritmo tiene en cuenta que en memoria se encuentran almacenados en orden inverso al que se han de mostrar, además de que no tienen porque existir siempre los tres números. Por ello, el algoritmo compara tanto existencia como numero de ellos, para así mostrarlos unívocamente en la pantalla.

```
formulario.show();
```

Esta clase también implementa la interfaz ActionListener, utilizada únicamente para la gestión de eventos provocados por comandos.

En este caso se disponen de dos comandos, atrás que siempre estará visible, y guardar números, que aparecerá en el caso de estar situados en la pestaña de números.

Para la diferenciación de la fuente se utilizan los identificadores de comandos. La funcionalidad del comando atrás es crear un nuevo objeto MenuCaptura, mientras que el comando guardar es el encargado de gestionar la validez y almacenamiento de los números en memoria.

Comando Guardar [ver Figura 54]

Como se indica en el apartado de análisis de la aplicación, para que un número se guarde tiene que ser de 9 dígitos, diferente al resto, y que el anterior a él exista, exceptuando el caso en que solo exista el primero. Para ello, se ejecuta un algoritmo que mediante variables booleanas y Strings contemplan todos los casos.

Variable	False/null	True/!null
Fallo	Cuando cualquier número no es válido.	Cuando el primer número es válido aunque el resto no lo sean.
Num1val	El primer número no es válido por tamaño o por coincidencia con otro	El primer número es válido y no coincide con ninguno de los restantes
Num2val	El segundo número no es válido por tamaño o por coincidencia con otro	El primer número es válido y no coincide con ninguno de los restantes
Num3val	El tercer número no es válido por tamaño o por coincidencia con otro	El primer número es válido y no coincide con ninguno de los restantes
mostrarDialogo	Los números son válidos por cuestiones de tamaño	Cuando cualquier número no es válido por cuestiones de tamaño.
Telefono1	Cuando no existe el primer número o no tiene tamaño adecuado	El primer número es válido por tamaño, toma el valor del cuadro de texto
Telefono2	Cuando no existe el segundo número o no tiene tamaño adecuado	El segundo número es válido por tamaño, toma el valor del cuadro de texto
Telefono3	Cuando no existe el tercer número o no tiene tamaño adecuado	El tercer número es válido por tamaño, toma el valor del cuadro de texto

Tabla 4: Significado del valor de las variables de control de validez de números introducidos

En primer lugar se obtienen los números del cuadro de texto. Tras obtenerlos tendremos actualizadas todas las variables, en cuanto a existencia y validez de tamaño.

Si el primer teléfono existe y es válido (telefono1!=null), la variable fallo pasara a false. Así, en el caso de que sólo exista él, finalmente se

guardará. En caso de que existan más, hasta que no se realicen todas las comprobaciones no se podrán guardar.

Si llegados a este punto, la variable mostrarDialogo está a true, indicará que alguno de los números existentes, no cumple con la característica de 9 dígitos, por lo que no hay que seguir con más comprobaciones, se mostrará un Dialogó de error (Dialog.TYPE_ERROR) indicando que alguno de los números existentes no cumple con la longitud deseada.

En caso contrario, se entiende que todos los números existentes cumplen el requisito de tamaño, por lo que se puede seguir el proceso.

En este punto, nuevamente hay que comprobar la existencia del primer número, si no existe, saltara un Dialogo de error indicando que falta el primer número.

Si existe, se pasará a la comprobación de igualdad entre los números. Si se produce alguna coincidencia se mostrara un dialogo error indicando que existen números iguales. La comprobación se realiza mediante el método equals de la clase String: `primero.equals(segundo)`

Por el contrario, si la comprobación es adecuada, ya estamos en disposición de guardar los números en memoria. Se almacenarán aquellos que sean válidos (num*X*Val=true). Con esto nos aseguraremos, que todos los números que existan se guardarán, aunque los siguientes no se guarden, porque no existen. Además, de que ninguno se guardará si su anterior no existe, por ejemplo, si num2val=false, el tercer número no se guardará aunque sea válido.

El proceso para guardar los números es similar al almacenamiento de palabras en lo que a lógica de apertura y cierre se refiere.

Una vez guardados de manera satisfactoria aparece un dialogo informativo `(Dialog.TYPE_CONFIRMATION)`, mediante el cual se confirma que el proceso se ha saldado con éxito.

Antes de terminar el proceso, con el fin de limpiar la pantalla de números erróneos, se comprobarán los String telefono*X*, limpiando el cuadro de texto asociado a los que sean null.

```
if (telefono2 == null) {
      numero2.setText("");
}
```

4.1.6. Menú Inicio

Esta clase se encarga de preguntar al usuario si desea que la aplicación se conecte al servidor a través de Internet, y actuar en base a la contestación.

```
public MenuInicio(Captura midlet_, final ConexionSms conexionSms)
```

Recibe los mismos parámetros que la clase ayuda y actuará de forma idéntica con ellos, los pasa a la siguiente pantalla sin actuar sobre ellos.

El cuerpo fundamental de la clase, sigue los siguientes pasos:

```
formulario=new Form("Bienvenido a SpyMode");
```

Creación del formulario sin más características especiales que el propio nombre.

```
añadirCampo();
```

Crea el cuadro de texto en el que se introduce el número, solo permite caracteres numéricos.

```
añadirExplicacion();
```

Crea el área de texto no modificable en el cual aparece el texto explicativo sobre la pantalla en la que nos encontramos

```
añadirComponentes();
```

Añade los comandos y componentes en el orden que se quiere que aparezcan así como el escuchador de comandos por defecto.

```
darEstilo();

formulario.show();
```

La clase implementa el interfaz ActionListener y es en el método actionPerformed donde se encuentra la funcionalidad real de la misma.

Así al pulsar el comando no, se establecerá la variable booleana Internet a false. Esta variable es declarada static (estática) y public (pública) por lo que será accesible desde cualquier clase de la aplicación y mantendrá su valor, salvo que se modifique específicamente.

Mientras que al pulsar el comando Si [ver Figura 52], arranca un algoritmo de comunicación y comprobación con el servidor, para que si todo ocurre satisfactoriamente, la variable Internet sea true, y en los momentos que a lo largo de la aplicación sea necesario el envío de datos al servidor, éste se produzca.

Para establecer la conexión con el servidor es necesario crear un objeto de tipo HttpConnection en base a una url `hc = (HttpConnection) Connector.open(url);`. así como indicar el tipo de método a utilizar (get o post) y añadir los parámetros que sean necesarios para la comunicación:

```
hc.setRequestMethod(HttpConnection.GET);
hc.setRequestProperty("telefono", telf);
```

Finalmente, para comprobar la validez de la conexión, es necesario leer la respuesta enviada por el servidor hc.getResponseCode() y en función del código actuar en consecuencia.

Si se produce algún tipo de error, bien al crear la conexión, o el indicado por la lectura de la respuesta, la manera de indicarlo al usuario será mediante la creación de un área de texto no modificable, que será incluida `formulario.addComponent(error);` inmediatamente debajo del cuadro de texto de número, con el texto asociado al error.

Antes de correr el algoritmo de conexión, cabe destacar, que si el numero introducido no es de 9 dígitos `if (telefono != null && telefono.length() == 9)`, aparecerá un dialogo de error `(Dialog.TYPR_ERROR)`.

Si el número es válido y se ha producido la comunicación con el servidor de manera exitosa, el siguiente paso será guardar el número de teléfono en la memoria (similar al resto de almacenamientos en memoria), para que así, cuando sea necesario, obtenerlo.

Finalmente, se crea un objeto de tipo MenuCaptura con el que se visualiza el siguiente nivel.

4.1.7. PantallaVideo

Esta clase se encarga de tomar las imágenes y aplicar el algoritmo de detección. En caso de producirse un movimiento, envía la imagen mediante los mecanismos que corresponda.

```
public PantallaVideo(Captura midlet_, ConexionSms conexionSms)
```

Los parámetros que recibe esta clase son el midlet que gobierna la aplicación así como el objeto de escucha de mensajes de texto, en ambos casos, no se realiza ninguna acción sobre ellos, tan solo se transportan hacia los diferentes destinos.

Se crea un formulario como base de la pantalla, aunque el peso de la visualización la lleva un objeto MediaComponent y la funcionalidad está gobernada por dos hilos diferenciados: uno de captura de imágenes (CameraThread) y otro de detección de movimiento (CapturarThread).

El hilo encargado de la cámara es el que crea el mediaComponet, sobre el cual estará alojado el player encargado del acceso y visualización de la cámara.

Para la creación del player hay que tener en cuenta qué es lo que se quiere capturar, y a éste hecho están totalmente ligadas las capacidades del terminal. Si se instancia un player con una característica que no soporta el terminal, evidentemente, no funcionará.

En nuestro caso, existen diferentes versiones tratando de ampliar la gama de terminales en los que funcione la aplicación sin ningún tipo de problemas. Por tanto, la creación del player será:

- `Manager.createPleyer("capture//video")`, para móviles que permitan el tomar imágenes a partir de un video.

- `Manager.createPayer("capture//image")`, para aquellos que no soporten la toma de imágenes de un vídeo, y se tenga que capturar por imágenes fijas.

Este hecho afecta directamente al rendimiento de la aplicación, puesto que la toma de imágenes consecutivas en un vídeo es más rápido que acceder a una imagen, guardarla y acceder a la siguiente.

Una vez se tiene el player, es necesario obtener su controlador (1) de vídeo, puesto que con él podremos tomar imágenes de la cámara. Se trata por tanto del elemento que tiene el control sobre los accesos a la cámara.

Un MediaComponent, sirve de contenedor para un player. En LWUIT se utiliza para la visualización de la cámara sin tener que acceder al canvas como se realiza en general. Con este componente, podemos actuar como con cualquier contenedor, lo que hace totalmente transparente el uso del player, puesto que además de sus características particulares, le podemos utilizar como cualquier otro componente en lo que a visualización se refiere.

Dicho mediaComponent, estará asociado con un player (2), sobre el cual podrá actuar para iniciarlo o pararlo, al iniciar o parar el mediaComponent también estaremos afectando al player que está asociado a él.

Como cualquier elemento que queramos incluir en la pantalla, el mediaComponente, hay que añadirlo al formulario (3), y el formulario ponerlo visible (4).

Finalmente cuando está estabilizado el sistema, con todos los elementos visuales creados y colocados, se inicia el mediaComponent (5), para así comenzar a ver las imágenes o el vídeo que toma la cámara.

```
(1)   videoControl = (VideoControl)
      player.getControl("VideoControl");
(2)    contendorPlayer = new
       MediaComponent(player);
(3)   formulario.addComponent
      (BorderLayout.CENTER, contendorPlayer);
(4)   formulario.show();
(5)   contendorPlayer.start();
```

Con este hilo, ya tenemos iniciada la cámara y podremos tener acceso a ella en cualquier momento, hasta que se pare el mediaComponent, y por tanto el player que aloja en su interior.

Mediante el otro hilo, CapturaThread, se realiza la obtención de imágenes y se ejecuta el algoritmo de detección de movimiento sobre ellas.

Una vez iniciado (1), para poder realizar la pausa que permite la colocación del dispositivo (ver…), se incluye una parada en el hilo (2).

```
hilocaptura.start();
hilocaptura.sleep(1000);
```

Dentro del método run de este hilo se sitúan los métodos encargados de la captura y detección. Para ello, lo primero que se hará, será tomar imágenes, de dos en dos, y aplicar el algoritmo por cada grupo de ellas.

El cuerpo principal se encuentra en el método tratamiento, a través del cual se realizan todas las acciones para cumplir la funcionalidad.

```
public void tratamiento() {

        Image[] imgdetect = capturar();

        Image prueba = imgdetect[0];
        Image prueba2 = imgdetect[1];

        int alto = prueba2.getHeight();
        int ancho = prueba2.getWidth();

        int[] rgbData = new int[ancho * alto];
        prueba.getRGB(rgbData, 0, ancho, 0, 0,
      ancho, alto);

        int[] rgbData2 = new int[ancho * alto];
```

```
                prueba2.getRGB(rgbData2, 0, ancho, 0, 0,
            ancho, alto);

                detección detección = new
            detección(rgbData, rgbData2);

                if (deteccion.hayDeteccion()) {

                    mandarImagen(imgdetect[1]);
                    if (MenuInicio.internet) {
                        mandarImagenHttp();
                    }
                    try {
                        hilocaptura.sleep(2000);
                    } catch (InterruptedException e) {

                        e.printStackTrace();
                    }
                }
            }
```

El método captura, devuelve un array con dos imágenes consecutivas, dichas imágenes se obtienen a través del VideoControl:

```
imageArray = videoControl.getSnapshot(null);
image      =      Image.createImage(imageArray,      0,
imageArray.length);
imagenes[i] = image;
```

Primero se obtiene el array con el getSnapshot, el parámetro null indica que no se especifica en ningún momento el formato de la imagen que se quiere tomar, así queda como responsabilidad de la cámara y de su controlador por defecto resolver este apartado, y evita numerosos problemas de incompatibilidades de formatos. Esta aplicación no necesita saber la codificación, puesto que trabaja a nivel de bytes con las imágenes, y además posteriormente sí realiza una codificación para el envío, pero solo con la intención de reducir el tamaño de la imagen. Finalmente crea la imagen y la guarda en el array de imágenes que va a devolver (dos imágenes).

Una vez se tienen las imágenes, se realiza una conversión a un array de enteros con información de color, puesto que el algoritmo utilizado se basa en la comparación de colores en distintos puntos de una misma imagen.

Cuando se tienen las imágenes en RGB, se aplica el algoritmo de detección, de ello se encarga la clase detección, anteriormente descrita.

Tras aplicar el algoritmo, podemos comprobar si existe o no detección, y en caso afirmativo, iniciar el proceso de envío. La imagen a enviar siempre será la segunda (un array comienza a numerarse en la posición 0), puesto que es en ésta donde se ha detectado el movimiento. Mandar la imagen, consiste en crear un objeto de la clase EnvioMms.

Además si al principio de la aplicación hemos dictaminado que se puede realizar la conexión con el servidor, también enviaremos esta imagen a través de una conexión http.

Finalmente, para evitar que sobre un mismo movimiento se realicen numerosas capturas, y por tanto se envíe un número excesivo de mensajes multimedia, también se para durante un tiempo el hilo de captura. Se entiende que con una imagen por detección de movimiento es suficiente, además de que un objeto describe un movimiento continuo sobre el campo visual de la cámara, y por tanto afectará a varios puntos de detección del algoritmo, y con que se produzca el movimiento sobre un solo punto ya se ejecuta las instrucciones de envío.

Esta clase implementa la interfaz ActionListener, para la escucha de eventos de comando, en este caso solo dispone de un comando atrás. Este comando al ser pulsado, deberá parar la detección e indicar que se sale de la pantalla al midlet, y finalmente arrancar la pantalla del menú anterior.

```
            case ATRAS:

                iniciar = false;
                midlet.setPantallaVideo(null);
                contendorPlayer.stop();

                releaseResources();

                new MenuCaptura(midlet, conexionSms);

                break;
```

Se ha incluido este comando para accesos erróneo a la captura de imágenes, para que durante el periodo de seguridad que protege el inicio innecesario de la detección, se vuelva a la pantalla anterior. Utilizar este comando para finalizar la captura con la aplicación en marcha, supone que, sí se pasa por el campo de visión de la cámara, se detectara un movimiento con el envío de mensajes asociado.

4.2. Receptor

4.2.1. Ayuda

En esta clase se crea el formulario sobre el que se incluyen los elementos visuales, la disposición en este caso consiste en un grupo de pestañas cuya navegación se sitúa en la parte izquierda y el contenido en la parte derecha.

```
public AyudaReceptor(Captura midlet_, final ConexionMms conexionMms)
```

En su constructor recibe dos parámetros, sobre ellos se actúa de manera similar a la clase AyudaCaptura, es decir se transportan por la pantalla hacia las siguientes. En este caso, además del midlet, se tiene un objeto de la clase ConexionMms, el cual es el encargado de la gestión de los mensajes multimedia que lleguen a la aplicación.

El cuerpo fundamental de la clase, sigue los siguientes pasos:

```
crearFormulario();
```

Se crea el formulario con las particularidades de que las transiciones de entrada y salida son de tipo cubo y no se permite el desplazamiento ni horizontal ni vertical.

```
crearPestañas();
```

Se crea el contenedor de pestañas cuya navegación está situada en la zona izquierda `pestañas = new TabbedPane(Component.LEFT);`, se indica la transición entre el pestañas, de tipo cubo vertical, bajo el mismo supuesto que la clase AyudaCaptura.

Dentro del método crearComponentesPestañas();, se crean los contenedores correspondientes a cada una de las pestañas, estos contenedores son: configuración, iniciar, pausa y terminar. Todos ellos incluirán un área de texto donde se situara el texto explicativo de la opción, y no permitirán el desplazamiento horizontal pero si el vertical por si la altura del texto es mayor que la de la pantalla.

```
darEstilo();

añadirComponentes();
```

Se añaden los componentes en el orden que se quiere que se visualicen, además del comando atrás, y de indicar que el escuchador de eventos de comandos es el por defecto de la clase

```
formulario.show();
```

La clase implementa la interfaz ActionListener, y en su método actionPerformed atenderá a los eventos generados dentro de la clase, en este caso, el

único evento contemplado es el del comando atrás, que cuando es pulsado, creara un nuevo objeto de la clase Menureceptor para su visualización.

4.2.2. Menú Receptor

Esta clase es el menú principal de la aplicación de recepción donde estarán incluidos los botones para navegar por las diferentes funcionalidades de las mismas: Configuración, iniciar aplicación, pausar aplicación, terminar aplicación y ayuda.

```
public MenuReceptor(Receptor midlet_, final ConexionMms conexionMms)
```

Recibe los mismos parámetros de la clase AyudaReceptor, también los transporta al resto de funcionalidades que nacen desde esta pantalla y además, utiliza el midlet para llamar a su método destroyApp cuando se pulsa el comando salir.

El cuerpo fundamental de la clase, sigue los siguientes pasos:

```
cliente = new BluetoothLocal();
```

Mediante este objeto se tiene acceso a la comunicación bluetooth para cuando sea necesario enviar un mensaje al dispositivo donde esté instalada la aplicación de captura y recepción.

```
crearFormulario();
```

Se crea el formulario con la característica de que su layout es de tipo BorderLayout[3].

[3] *Se utilizan cinco zonas para colocar los componentes: Norte, Sur, Este, Oeste y Centro.[ver Figura 14]*

```
crearBotonera();
```

Se crea un contenedor sobre el cual se incluirán el resto de botones.

```
crearBotones();
```

Se crean los botones que aparecerán en la pantalla, una vez creados se les añaden sus escuchadores de eventos (addActionListener), para dictaminar la acción a realizar cuando son pulsados.

Configuración

Al pulsar este botón se crea un nuevo objeto de tipo menuSeguridad, en el cual poder realizar la configuración de la aplicación tanto en temas de seguridad (palabras clave o contraseñas), como el número de envío para el funcionamiento remoto. `new MenuSeguridad(midlet, conexionMms);`

Iniciar aplicación [ver Figura 55]

Al pulsar este botón, lo primero que hará la aplicación es tratar de abrir el RecordStore donde se encuentra la palabra de inicio `RecordStore.openRecordStore("Inicio", false);`, con este método, si no existe un recordStore con el nombre indicado en el primer parámetro no creará uno nuevo.

En caso de que no exista el RecordStore, saltará un mensaje de error `(Dialog.TYPE_ERROR)`, indicando que no hay palabra de inicio en memoria.

```
catch (RecordStoreNotFoundException e) {
dialogoFaltaPalabra();
}
```

Si existe el recordStore, se creará un String con el contenido y se cerrara el recordStore.

Si el proceso ha sido correcto la variable enviarInicio estará a true, y por tanto se permite el envío de un mensaje de texto con la palabra que se encuentra en el RecordStore de Inicio.

```
iniciar.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent evt) {

String palabraInicio = obtenerpalabraInicioSms();
            if (enviarInicio) {
                  envioSms = new
            EnvioSms(palabraInicio + "iniciar",
            midlet, conexionMms);
            envioSms.start();
            }

}
});
```

El objeto envioSms es un hilo, por lo que además de crearlo, hay que llamar a su método start para que inicie su funcionalidad.

Pausar aplicación [ver Figura 55]

Se actúa de la misma manera que para el caso de iniciar aplicación solo que el recordStore sobre el que se trabajara será "Pausa", y su variable de control será enviarPausa. Utilizará también un objeto de la clase envioSms para el envío del mensaje de texto al otro terminal.

Terminar aplicación [ver Figura 56]

Cuando se pulsa este botón, se cede el control a un diálogo para que el usuario decida el tipo de finalización que desea:

```
terminar.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
      dialogoTerminar();
}});
```

En dicho diálogo aparecerán dos RadioButton incluidos en un ButtonGroup. Un radioButton permite realizar una selección, mientras que al estar incluidos en un ButtonGroup, la selección es única, es decir, solo puede tener un elemento seleccionado.

Los botones se crean con su texto asociado `terminarSms = new RadioButton("Via sms");` `terminarBlue = new RadioButton("Via bluetooth");`y se inlcuyen mediante el método add, `smsBlue.add(terminarBlue);smsBlue.add(terminarSms);`al Buttongruop, creado anteriormente `smsBlue = new ButtonGroup();`

El buttonGroup no realiza ninguna muestra visual de los botones, sirve para controlar su funcionamiento, por lo que los RadioButton también tienen que ser introducidos en la pantalla o contenedor donde se vaya a mostrar. En este caso, en el dialogo se incluirá un contenedor sobre el cual estarán incluidos el resto de elementos. La particularidad de este contenedor es que sus elementos están dispuestos en base a un boxLayout en Y[4] `new Container (new BoxLayout (BoxLayout.Y_AXIS));`.

El dialogo, como ya se ha dicho funciona como un formulario, por lo que los elementos hay que añadirlos en el orden que se quieren mostrar, así también dispone de la posibilidad de tener comandos, cuyo escuchador, se puede asociar al escuchador por defecto de la clase o a uno especifico, en este caso se utiliza uno particular `terminarAplicacion.setCommandListener (escucharComandoDialogo());`.

Dentro de este escuchador se atenderá única y exclusivamente a los comandos del diálogo (“Si” y “No”).

En caso de que el comando pulsado sea no, simplemente se eliminará el dialogo y aparecerá la pantalla del menuReceptor.

[4] *De forma vertical [ver Figura 14]*

Por el contrario, al pulsar el botón si, se comprobara qué elemento del botonGroup esta seleccionado (siempre habrá uno seleccionado) `smsBlue.getSelectedIndex()`, y se actuará en consecuencia.

Si el elemento seleccionado es el de terminar vía bluetooth, con el objeto cliente se llamará al método que realiza los pasos para enviar los datos para terminar la aplicación.

```
if (smsBlue.getSelectedIndex() == 0) {
      cliente.buscarDispositivosRemotos();

}
```

En cambio, si se desea terminar la aplicación vía SMS, se seguirán los mismos pasos que en los botones iniciar y pausar aplicación, tan solo que se atacará al recorStore Fin y su variable de control será enviarFin.

```
else if (smsBlue.getSelectedIndex() == 1) {
      String palabrafin = obtenerpalabraFinSms();
            if (enviarFin) {
envioSms = new EnvioSms(palabrafin + "terminar",
            midlet, conexionMms);
envioSms.start();
                  }
            }
```

Cabe destacar que en los envíos, se introduce un String adicional a la palabra clave, con el fin de distinguir su procedencia en el caso de que se utilice la misma palabra.

Ayuda

Al pulsar este botón se crea el objeto AyudaCaptura que muestra la pantalla con los textos explicativos sobre cómo funciona la aplicación.

```
ayuda.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
new AyudaReceptor(midlet, conexionMms);
}});
```

```
darEstilo();

añadirComponentes();
```

Se añaden los botones al contenedor botonera, en el orden que van a aparecer, y dicho contenedor se incluye en el formulario principal, además se añade el comando salir y el escuchador de comandos por defecto

```
formulario.show();
```

Esta clase implementa la interfaz ActionListener, y en su método actionPerformed, únicamente atiende a los eventos generados por el comando salir (los comandos de diálogo son atendidos por otro objeto ActionListener). En caso de que pulsemos el botón salir, se cerrará la aplicación.

```
if (c.getId() == SALIR) {
      midlet.destroyApp(true);
      midlet.notifyDestroyed();
}
```

4.2.3. Menú Seguridad

Esta clase se encarga de mostrar y gestionar las opciones de seguridad, es decir, las palabras clave o contraseñas, así como el número de teléfono del terminal donde se encuentra instalada la aplicación de captura y detección, para así poder realizar la configuración y uso remoto.

```
public MenuSeguridad(Receptor midlet_, ConexionMms conexionMms)
```

Los parámetros así como la utilización de los mismos es igual a la clase de ayudaCaptura.

El cuerpo fundamental de la clase, sigue los siguientes pasos:

```
crearFormulario();
```

Se crea el formulario con las transiciones de entrada y salida de tipo cubo horizontal así como con la particularidad de no permitir el desplazamiento horizontal ni vertical.

```
crearAreasTexto();
```

Crea los cuadros de texto que se van a utilizar para introducir las palabras claves `(setConstraint(TextArea.PASSWORD);)` y el número de envío `(setConstraint(TextArea.NUMERIC); setInputModeOrder(new String[] { "123" });)`

```
crearPestañas();
```

La pantalla consta de un único contenedor de pestañas en el que se incluirán los diferentes contenedores donde están situados los elementos de cada una de las pestañas.

Dicho contenedor tendrá su navegación en la parte superior `(new TabbedPane(Component.TOP);)` y no permitirá el desplazamiento horizontal ni vertical.

Dentro de la pestaña del numero de envío se encuentra un titulo indicativo de la pestaña `Label titulo = new Label("Introduzca el numero del otro terminal");`, un cuadro de texto donde introducir el numero de envío y un botón que permite guardar el contenido del cuadro de texto en memoria, siempre y cuando este sea válido.

Cuando pulsamos el botón, el evento es recogido por el escuchador específico del mismo [ver Figura 57].

En respuesta al evento, se obtendrá el numero introducido en el cuadro de texto `(numeroString = numeroEnvio.getText();)`, se comprobará su

validez (`if(num.length()==9)`) y en caso correcto, se procederá a guardar en memoria realizándolo de idéntica manera al resto de almacenamientos de la aplicación.

Antes de guardar el número, se añade el prefijo +34 por código, para que sea posible utilizarlo como numero de envío de mensajes de texto, ya que es un requisito imprescindible.

Las pestañas de inicio, pausa y fin son idénticas entre si, en lo que a aspecto y funcionalidad se refiere, con la salvedad de que cada una atiende a la gestión de una palabra diferente [ver Figura 58].

Estas pestañas estarán compuestas por un título en la parte superior, un cuadro de texto donde introducir la palabra deseada, y en la parte inferior, un contendor donde se alojan dos botones cuya funcionalidad está separada en sus escuchadores de eventos particulares.

El botón confirmar datos, obtiene la palabra del cuadro de texto `(getText())` y realiza la comprobación de existencia `(if (palabra != null && palabra.length() > 0))`

Si el resultado es false, termina la ejecución con un dialog `(Diagog.TYPE_ERROR)`, en el que indica que no existe ninguna palabra.

Por el contrario si la palabra es válida, se procede a guardar en memoria, como el resto de almacenamientos.

Una vez está en la memoria permanente del teléfono, aparece un dialogo de confirmacion `(Dialog. TYPE_CONFIRMATION).`

Adicionalmente, cuando desaparece el dialogo de confirmación, aparece uno nuevo en el que se pregunta al usuario si se quiere o no enviar al otro terminal. Este dialogo, poseerá sus propios comandos, que serán creados específicamente para cada uno de ellos. Sus nombres serán iguales, pero sus

identificadores variarán con el fin de que con un mismo escuchador de eventos se puedan gestionar las acciones a realizar cuando se pulsan.

El comando "Si" obtendrá los valores 1, 3 y 5 respectivamente para inicio, pausa y fin, siendo el valor del identificador de los comandos "No" 2, 4 y 6 para los mismos casos.

La diferenciación entre comandos dentro del escuchador estará controlada con una estructura switch-case, en la que se evalúa una función `(evt.getCommand().getID())` y se establecen una serie de casos, 1, 3 y 5, para el supuesto que nos ocupa.

La reacción ante el comando "Si", será siempre la misma, enviar un mensaje de texto, lo que variara será el contenido de dicho mensaje según su procedencia, se incluirán así, las palabras de inicio (1), la de pausa (3) o la de fin (5).

El botón guardar y enviar realiza en esencia la misma funcionalidad que el botón confirmar si en el diálogo se acepta el envío. Solo que en este caso se realiza un proceso continuo sin preguntar al usuario, y la única información que se le va es cuando termine todo el proceso de manera satisfactoria.

Así, se obtendrá la palabra, se guardará en memoria y se enviara el mensaje correspondiente, de forma transparente al usuario.

La pestaña envío, consta de un único botón central `enviarDatos.setAlignment(Button.CENTER);` cuya misión consistirá en acceder a cada uno de los recorStore de las palabras, para obtener su valor, e incluirlo en un envío de mensaje de texto, en este caso no se realiza ninguna comprobación sobre la palabra, puesto que si esta en memoria se supone que es porque el usuario la ha introducido y existe, sino el RecordStore estará vacío y no se enviara ninguna palabra. Es responsabilidad del usuario el rellenar los campos antes de enviarlos.

La pestaña de instrucciones consta de un contendor en el que se incluye un cuadro de texto plano no editable con desplazamiento vertical.

```
darEstilo();

ponerNumeroInicial();
```

Como ocurría en el caso de la pestaña de número situada en el menú de configuración de la aplicación captura, en este caso, si existe en memoria el número de envío, se mostrará en el cuadro de texto cuando se acceda a la pestaña. Esto se realiza abriendo la memoria, y obteniendo el valor en caso de que exista `numeroEnvio.setText(str.substring(3, str.length()));`. El número 3 del primer parámetro indica que se empiece en la tercera posición, así, eliminaremos el prefijo +34 de la visualización.

```
añadirComponentes();
```

Se añaden los contenedores al contenedor de pestañas en el orden en que van a aparecer: numero, inicio, pausa, fin, envío e info; así como el comando atrás y se registra el escuchador por defecto de comandos (commandListener)

```
formulario.show();
```

Esta clase implementa el interfaz ActionListener, y en su método actionPerformed atenderá al comando atrás, cuya funcionalidad está en mostrar la pantalla anterior, MenuReceptor

El resto de comandos, los situados en los diálogos, poseen su propio escuchador.

4.2.4. Receptor

Es el Midlet de la aplicación, y como tal, es la primera clase que será llamada al arrancar la aplicación.

```
public Receptor(){
```

En su constructor únicamente se creará el objeto conexiónMms, mediante el cual, se realizara la atención a los mensajes multimedia que lleguen al terminal móvil mientras este activa la aplicación (poner referencia a conexionMms).

```
}
```

Como todo midlet, ha de implementar los tres métodos del ciclo de vida:

```
public void startApp(){
```

Es el método mediante el cual se arranca el hilo de ejecución de la aplicación.

Activará el apartado visual de lwuit Display.init(this); asociado al midlet, y posteriormente creará la primera pantalla de visualización (`new MenuReceptor(this, conexionMms);`)

```
}

public void pauseApp(){
```

Es el método utilizado para pausar la aplicación cuando así se indique. No se realiza ninguna acción, puesto que dentro del desarrollo de la aplicación no se tiene contemplado ningún caso en el que se tenga que pausar la aplicación.

```
}
```

```
public void destroyApp(boolean incondicional) {
```

Este es el método utilizado para indicar que pasos hay que realizar cuando se cierre la aplicación, en este caso, el único requisito es que se abandone, así que no es necesario implementar ninguna funcionalidad en el.

```
}
```

Como particularidad, cabe destacar el hecho de que la conexión MMS no se cree dentro del hilo de ejecución principal, esto es debido a que el hilo de atención a mensajes multimedia es independiente al hilo principal de la aplicación y debe estar siempre activo y circular por la aplicación sin tener en cuenta el estado del midlet.

4.3. Comunicaciones

4.3.1. Mensajería

4.3.1.1. Conexión SMS

Esta clase es la encargada de la escucha activa de mensajes de texto.

```
public ConexionSms(Captura midlet)
```

Recibe como parámetro un objeto de tipo Captura, sobre el cual no realiza ninguna acción, simplemente lo traslada al resto de clases.

En esta clase se abre la conexión para poder recibir mensajes de texto, para abrir la conexión se utiliza `conexion = (MessageConnection) Connector.open(`direccionSms);

La variable que incluimos al crear el conector debe tener la siguiente forma: `"sms://:" + PUERTO_SMS` siendo el puerto el 5000. Así indicamos que la conexión que deseamos es de SMS en el puerto 5000.

Además para que se actúe cuando llega un mensaje es necesario registrar un escuchador de eventos de llegada de menajes `conexion.setMessageListener(this);`

Esta clase implementa la interfaz MessageListener, que actúa de manera similar a la clase ActionListener, solo que en este caso los eventos que gestiona son los de llegada de mensajes.

Para esta gestión, posee un método al cual recurrirá cuando se produce algún evento asociado a la mensajería en esta conexión. `public void notifyIncomingMessage(MessageConnection con)`

Dentro de este método se deben incluir los pasos a seguir que se desean realizar cuando llega un mensaje de texto.

Para evitar posibles bloqueos de la aplicación cuando se recibe un mensaje, la rutina de atención a mensajes se tiene que realizar en un hilo diferente al de ejecución, para ello, dentro de esta clase se utiliza la clase interna HiloSms.

Esta clase interna implementa la interfaz Runable (es un hilo), y por tanto debe sobrescribir el método run, el cual será llamado cuando se estime necesario el inicio del hilo.

Entonces, al recibir un mensaje, el método notifyIncomingMessage iniciará el hilo de escucha:

```
public void NotifyIncomingMessage(MessageConnection con) {
        enEjecucion = true;
        hiloSms = new HiloSms();
        hiloSms.run();

}
```

La variable enEjecución, determinará en qué momento se ha de obtener el mensaje. La conexión siempre está escuchando, pero el hilo solo debe

intentar acceder a un mensaje cuando éste llegue, si intenta obtener un mensaje de la conexión cuando este no existe se producirá un error grave en la ejecución, y es un caso no deseable.

Dentro del método run del hilo, se comprobará que realmente se está escuchando (variable enEjecución), se obtendrá el mensaje (1) y se llamará a la clase RecibirSms(4), que es la encargada de decodificar el mensaje y realizar la acción oportuna.

Para evitar problemas, aunque sea una acción redundante, porque se supone que si hemos iniciado el hilo es porque ha llegado un mensaje, es necesario comprobar la validez del mensaje (2), y una vez es válido, ciertamente solo interesan los mensajes de texto, por lo que se comprueba el tipo de mensaje y se descartan los que no sean de texto (3).

```
public void run() {

   while (enEjecucion) {

      try {
      (1)   mensajeRecibido = conexion.receive();

      (2)   if (mensajeRecibido != null) {

      (3)   if (mensajeRecibido instanceof TextMessage) {

      (4)   new RecibirSms(mensajeRecibido, midlet);

}
```

Además, dentro de esta clase, se encuentra un método de atención a mensajes de texto para cerrar la conexión, necesitaremos cerrarla cuando la aplicación se disponga a enviar un mensaje multimedia, puesto que la convivencia de dos conexiones en paralelo, no está bien soportada en los terminales y genera grandes problemas de funcionamiento.

Para ello, se comprobará que existe la conexión (1), para no acceder a una conexión inexistente, se desregistra el escuchador de mensajes (2), así se devuelve el control de la escucha de mensajes a la aplicación que tiene el terminal destinada para ello y finalmente se cierra la conexión (3).

Para asegurar que está cerrada y evitar que el hilo no continúe su ejecución, se cambian los valores de conexión (4) y enEjecucion (5).

```
public void cerrarConexionSms() {
        (1)    if (conexion != null) {
                        try {
        (2)    conexion.setMessageListener(null);
        (3)    conexion.close();
                } catch (IOException e) {

                }
        (4)conexion = null;
        (5)enEjecucion = false;
        }

}
```

4.3.1.2. Conexión MMS

Esta clase es la encargada de la escucha activa de mensajes multimedia.

```
public ConexionMms(String appID)
```

Recibe como parámetro un string donde está el valor de la propiedad de mensajería multimedia que se asocia a la aplicación

```
String appID = getAppProperty("MMS-EnvioID");
```

El valor de esta propiedad se encuentra en el fichero .jad (java aplication descriptor) de la aplicación.

En base a dicha propiedad se crea la conexión de mensajería multimedia: `conexion = (MessageConnection) Connector.open(destino);`.

La variable que incluimos al crear el conector debe tener la siguiente forma: `destino = "mms://:" + appID;` siendo `appId`, el valor que le llega al constructor, que no es otro que el valor de la propiedad `("MMS-EnvioID");` del fichero .jad. Así se creará una conexión de mensajería multimedia asociada a la aplicación.

También implementa la interfaz MessageListener, así que su método NotifyIncomingMessage atenderá a la llegada de mensajes a la conexión.

```
public void NtifyIncomingMessage(MessageConnection con) {
            enEjecucionMms = true;
            hiloMms = new HiloMms();
            hiloMms.run();
}
```

Al igual que en la clase ConexionMms, la atención a los mensajes se desarrollará en un hilo aparte para evitar bloqueos, y dicho hilo será una clase interna.

Cuando llegue un mensaje realizará los mismos pasos que para un mensaje de texto, solo que en este caso la variable de control será enEjecucionMms, y el tipo de mensajes de interés serán aquellos que sean del tipo MultipartMessage.

```
public void run() {
    while (enEjecucionMms) {
      try {
       mensaje = conexion.receive();
       if (mensaje != null) {
            if (mensaje instanceof MultipartMessage) {
                  new RecibirMms(mensaje);
             }
        }
    }
}
```

También dispone de un método para cerrar conexiones, que trabaja de forma similar al de los mensajes de texto, con la misma funcionalidad.

```
public void cerrarConexionMms() {
        if (conexion != null) {
             try {
                    conexion.setMessageListener(null);
                    conexion.close();
                    conexion = null;
                    enEjecucionMms = false;
                    }
             catch (Exception e) {
             }
        }
}
```

4.3.1.3. Envío SMS

Esta clase se encarga del envío de mensajes de texto [ver Figura 63]

```
public EnvioSms(String textoEnvio, ConexionMms conexionMms)
```

Esta clase recibe como parámetros:

- textoEnvio. Se trata del String que va a ir incluido en el mensaje de texto.
- conexionMms. Es la conexión que gobierna la escucha de mensajes multimedia en la aplicación del receptor, es necesaria ya que se cerrara antes de realizar el envío del mensaje de texto, y se restablecerá cuando termine dicho envío.

La convivencia entre las conexiones de mensajes de texto y mensajes multimedia, se ha comprobado que es inviable, ya que produce incompatibilidades y accesos erróneos a la conexión para realizar el envío, o la escucha. Llegando incluso a

equivocarlas, por lo que para utilizar una, es imposible que la otra este activa y viceversa.

Extiende de la clase Thread, lo que implica que deberá implementar el método run, con el cual el hilo empezará la ejecución.

Dentro del constructor, la primera acción que se realiza es obtener el número de envío de la memoria permanente de teléfono, este acceso al recordStore se hace sin crearlo en caso de que no exista (), si se puede acceder y obtener, se permitirá el envío (la variable booleana enviar se encarga de ello), y en caso contrario aparecerá un mensaje de error (Dialog.TYPE_ERROR), y se interrumpirá el proceso.

Una vez obtenido el número se cierra el recordStore para evitar problemas en futuros accesos.

Dentro del método run está la llamada al método enviarSmsA(String destino), que es el realmente encargado de la gestión de las conexiones y el envío propiamente dicho.

Lo primero que hará será establecer el destinatario del mensaje (1), para ello utiliza el numero que se obtiene de la memoria y que se le ha pasado como parámetro. Cerrará la conexión de escucha de mensajes multimedia llamado a su método correspondiente (2). En este momento ya está en disposición de enviar el mensaje de texto, para ello se crea la conexión para el funcionamiento como SMS (3), se crea el mensaje indicando que es de tipo texto (4), se añade el destino (5) y el texto (6) y finalmente se envía (7).

Para volver a restablecer el funcionamiento de escucha de mensajes multimedia, se elimina la conexión que se ha creado (8), y se reabre la anteriormente existente (9).

```
public void enviarSmsA(String destino) {
        String destinoEnvio;
              (1)   destinoEnvio = "sms://" + destino + ":" +
        PUERTO_SMS;

              (2)   conexionMms.cerrarConexionMms();
                    try {
              (3)   conexion = (MessageConnection)
                 Connector.open(destinoEnvio);

              (4)   mensaje = (TextMessage) conexion
                    .newMessage(MessageConnection.TEXT_MESSAGE);
              (5)   mensaje.setAddress(destinoEnvio);
              (6)   mensaje.setPayloadText(textoEnvio);
              (7)   conexion.send(mensaje);
                    } catch (InterruptedIOException e) {
                     Dialog.show("InterruptedIOException",
                    e.getMessage(), "Aceptar",null);
                     } catch (IOException e) {
                     Dialog.show("IOException", e.getMessage(),
                      "Aceptar", null);
                                }
                     if (conexion != null) {
                          try {
                          (8)   conexion.close();
                          } catch (IOException e) {
                                Dialog.show("IOException",
                          e.getMessage(), "Aceptar", null);
                                }
                                conexion = null;
                                }

                          (9)   conexionMms.abrirConexionMms();

                          }
                     }
```

4.3.1.4. Envío MMS

Esta clase se encarga del envío del mensaje multimedia con la imagen [ver Figura 62].

```
public EnvioMms(Image imagenAConvertir, Captura midlet, ConexionSms
        conexionSms,PantallaVideo pantallaVideo)
```

Esta clase recibe como parámetros:

- imagenAConvertir: Se trata de un objeto de tipo Image. Sobre esta imagen se ha producido la detección de movimiento en la clase pantallaVideo, y es la que tendremos que enviar. Deberemos realizar una conversión de imágenes para que tenga un tamaño menor y sean prudenciales el número de mensajes multimedia necesarios para su envío (1-3 mensajes).

- Midlet: Se trata de un objeto de tipo Captura, es el midlet de la aplicación, es necesario para obtener la propiedad de envío para mensajes multimedia, indispensable para crear la conexión de envío multimedia.

- conexionSms: Es un objeto de tipo ConexionSms, es el encargado de la escucha activa de mensajes de texto a lo largo de la aplicación. Antes de realizar un envío multimedia, se cerrara; restableciéndola cuando este termine.

- pantallaVideo: Objeto PantallaVideo que se encarga de arrancar la cámara así como la detección de movimiento, se utilizarán sus métodos de control para pausar o iniciar la captura y detección según sea necesario.

Es una clase que extiende de Thread, por lo que es necesario implementar el método run () con el cual empezara la ejecución del hilo.

Antes de arrancar su funcionalidad, en el constructor realiza una serie de pasos:

Crea la imagen final de envío:

```
(1)  imagenParaEnviar = new
     ByteArrayOutputStream();
(2)  new  JpegEncoder(imagenAConvertir,  50,
     imagenParaEnviar);
```

Para ello dentro de un array de datos de salida (1), se introduce la imagen codificada a formato jpg.

La clase JpegEncoder, propiedad de ames R. Weeks and BioElectroMech, Copyright (c) 1998, de uso libre, se encarga de realizar la conversión de la imagen que toma la cámara a formato jpg con una calidad determinada por el segundo parámetro. El termino calidad se basa en el coeficiente de compresión, 0-100. De mayor calidad a menor o de mayor comprensión a menor.

Para continuar, se obtiene la propiedad de envío MMS del midlet (`appID = midlet.getAppProperty("MMS-EnvioID");`) y finalmente se accede a la memoria permanente del teléfono en busca de los números de envío.

Se abre el recordStore asociado a los números de envío (`numeros = RecordStore.openRecordStore("Numeros",true, RecordStore.AUTHMODE_PRIVATE, true);`), se comprueba si tiene algún tipo de contenido: `numeros.getNumRecords() > 0`. En caso de no existir ningún número, se parará la captura y detección, y aparecerá un Dialogo de error (Dialog.TYPE_ERROR), que bloqueará la aplicación hasta que el usuario acepte, momento en el que se arrancará nuevamente la aplicación. El hecho de parar por completo la aplicación es debido a que sin la posibilidad de envío, deja de cumplir una de sus funcionalidades básicas y por tanto no es necesario que siga ejecutándose.

Si tiene algún tipo de contenido; un, dos o tres números, se obtendrán y se guardarán en una variable temporal (`numero1 = new`

`String(numeros.getRecord(1));`, para posteriormente utilizarlos como destino de envío.

Ya el hilo de ejecución dentro del método run, únicamente hará la llamada al método de envío en el caso de que existan los números, añadiendo, previamente el prefijo "+34", para el envío:

```
if (numero1 != null) {
     numero1 = "+34" + numero1;
enviarMmsA(numero1);
}
```

Es el método `enviarMmsA (int destino)`, el encargado de la gestión de las conexiones, de la creación del mensaje multimedia, y finalmente de su envío.

Este método, en un primer momento creará la dirección de destino de la comunicación para el envío (1), utilizando el número de envío y la propiedad del teléfono asociada a tal fin.

Posteriormente iniciará la gestión de las conexiones existentes, con el fin de compaginarlas sin que estén activas en el mismo tiempo, para ello se deberá cerrar la conexión de escucha activa de mensajes de texto (2), y crear la nueva conexión de envío de mensajes multimedia (3).

Con la conexión creada, llega el momento de ocuparse del mensaje multimedia (3). Se trata de un mensaje Multipart que permite añadir diferentes elementos diferenciados en partes, para este caso, al enviar una sola imagen, solo se necesita una parte.

Para poder crear la parte (5), la imagen de envío debe ser un objeto de tipo InputStream, que se obtiene a partir de la imagen convertida (4).

Con todo ya creado, se añade la única parte al mensaje (6), así como su subject (7), que será el texto que tendrá asociado el mensaje, y servirá para distinguirlo del resto de mensajes multimedia en la aplicación de recepción.

Finalmente, se procede al envío propiamente dicho (8) y a reestablecer la situación de conexiones, cerrando la multimedia (10) y reabriendo la de texto (11).

```
        public void enviarMmsA(String destino) {
(1)     String destinoEnvio = "mms://" + destino + ":" + appID;
        try {
(2)     conexionSms.cerrarConexionSms();
(3)     conexion = (MessageConnection)Connector.open(destinoEnvio);
(4)     mensaje = (MultipartMessage) conexion
        .newMessage(MessageConnection.MULTIPART_MESSAGE);
(5)     InputStream is = new ByteArrayInputStream(
        ((ByteArrayOutputStream)imagenParaEnviar). toByteArray());
(6)     parte = new MessagePart(is, "img/jpg", "id-10", null,null);
(7)     mensaje.addMessagePart(parte);
(8)     mensaje.setSubject("Alerta Intruso");
(9)     conexion.send(mensaje);
        } catch (SizeExceededException e1) {
        Dialog.show("SizeExceededException", e1.getMessage(),
        "Aceptar",null);
        }
        if (conexion != null) {
        try {
(10)    conexion.close();
        } catch (IOException e) {
        Dialog.show("IOException", e.getMessage(), "Aceptar",
        null);
        }
        conexion = null;
        }
        try {
(11)    conexionSms.abrirConexionSms();
        } catch (Exception e) {
              Dialog.show("Exception", e.getMessage(), "Aceptar",
        null);}}
```

4.3.1.5. Recibir SMS

Una vez se recibe un mensaje de texto, esta clase se encarga de la gestión del mismo para decidir la acción a realizar [ver Figura 61].

```
public RecibirSms(Message mensaje, Captura midlet)
```

Este método recibe como parámetros, un objeto de tipo Message, que será el mensaje recibido por la aplicación y que cumple el requisito de que es instancia de TextMessage, y objeto captura, que es el midlet de la aplicación, necesario para, a través de sus métodos, realizar las acciones asociadas a cada tipo de mensaje.

Tras crear el mensaje de texto en base al recibido (`mensajeTexto = (TextMessage) mensaje;`) y obtener de él su texto (`entrada = mensajeTexto.getPayloadText();`). Se debe comprobar la longitud del texto, ya que el tratamiento variará en función de ella [ver Tabla 2].

Un mensaje especial contendrá un código que hará que el mensaje tenga más de 40 caracteres y un mensaje normal, su longitud no superara los 40 caracteres. Por mensaje especial se entiende aquel que afecta a la configuración de la aplicación (palabras claves) y por mensaje normal aquel que afecta la funcionalidad (inicio, pausa y fin).

```
if (entrada.length() > 40) {
    mensajeEspecial();
    } else {
    mensajeNormal();
    }
```

La atención a mensajes especiales, siempre se realiza de la misma manera, se lee el código identificativo (los 40 primeros caracteres) y se hace una comparación entre los distintos códigos y se actúa en base a ellos.

- Modificar palabra inicio
- Modificar palabra pausa
- Modificar palabra fin

Para estos tres primeros casos, la manera de actuar es idéntica. Se obtiene del mensaje la palabra que hay que introducir en memoria, es decir, se quita el código identificativo (`palabraInicio = entrada.substring(40, entrada.length());`). Y se pasa a la gestión de almacenamiento en la memoria: Se accede al RecordStore indicado, creándolo en caso de que no exista (`inicio = RecordStore.openRecordStore("Inicio",true,RecordStore.AUTHMODE_PRIVATE , true);`), así se cubre el caso en el que se configure por primera vez de manera remota.

Una vez creado, se comprueba si tiene o no algún contenido en su interior. Si tiene contenido, se sigue la política de cerrar, borrar, crear nuevamente, guardar y cerrar. Si no tiene contenido solamente se guardará y se cerrará.

- Modificar las tres palabras

Para este caso, hay que realizar el mismo proceso que se realiza para las palabras por separado, solo que de manera secuencial en un solo bloque de ejecución. Así, se obtienen las palabras, se guarda la de inicio (siguiendo los pasos de almacenamiento en memoria), posteriormente la de fin y para terminar la de pausa.

Mientras que para un mensaje normal se abren los recordStore de palabras y se obtienen las mimas. Para el caso en que las palabras sean coincidentes entre sí, se crea un mecanismo para dictaminar a que configuración se refiere, así a la palabra de inicio se le añade el String “iniciar”, a la de fin, “terminar” y a la de pausa “pausar”. Así aunque sean la misma palabra, se diferencia a que funcionalidad están afectando.

Una vez se obtienen las palabras, mediante el método acción se dictamina el proceso a seguir, y cuando este termine, se cierran los recordStore de las palabras, para evitar futuros conflictos de lectura.

```
public void mensajeNormal() {

        abrirRecordStore();
        stringInicio = obtenerString(inicio) + "iniciar";
        stringFin = obtenerString(fin) + "terminar";
        stringPausa = obtenerString(pausa) + "pausar";
        acción(entrada, stringInicio, stringFin, stringPausa);
        cerrarRecordStore();
}
```

El método acción, cuerpo de la funcionalidad ante la recepción de un mensaje normal, recibe como parámetros el texto extraído del SMS (entrada) y las tres palabras obtenidas de la memoria permanente del teléfono con su identificador añadido.

```
public void accion(String entrada, String stringInicio, String
stringFin,  String stringPausa) {
        if (entrada.equals(stringInicio)) {
              iniciarDeteccion();
        }
        if (entrada.equals(stringFin)) {
              finalizarAplicacion();
        }
        if (entrada.equals(stringPausa)) {
              midlet.pauseApp();
              }
        }
```

Compara dichas palabras con el contenido del mensaje y actúa en función del resultado.

- Para el caso en que coincide con la palabra almacenada en Inicio, el propósito de `iniciarDeteccion()`, es llamar al método asociado al midlet para el inicio (`midlet.iniciar()`).

- En el caso de finalizarAplicacion, se llama al método destroyApp del midlet.

- Si la coincidencia es con la palabra de pausa, se llama al método pauseApp del midlet.

4.3.1.6. Recibir MMS

Esta clase se encarga de crear y mostrar la imagen que se recibe en el mensaje multimedia [ver Figura 60].

```
public RecibirMms(Message mensaje)
```

Tiene como parámetro el mensaje que ha recibido la aplicación que es instancia de MultipartMessage.

De dicho mensaje se obtiene, primeramente el subject (1), que será utilizado como título, y posteriormente las partes (en este caso solo una) (2), a partir del cual se obtienen los datos (3) para crear la imagen (4). Con esta imagen (dentro de un Label) se crea un diálogo, poniéndole como título el subject y como objeto principal el Label que contiene la imagen.

Dicha imagen, es reescalada para que se adapte al tamaño del cuadro de dialogo (5)

```
public RecibirMms(Message mensaje) {
      mensajeMms = (MultipartMessage) mensaje;
      (1)   String subject = mensajeMms.getSubject();
      (2)   MessagePart[] partes = mensajeMms.getMessageParts();
            if (partes != null) {
                  for (int i = 0; i < partes.length; i++) {
                  MessagePart parteMms = partes[i];
                  (3)  recibido = parteMms.getContentAsStream();
                 }

            }
```

```
                Dialog dialogo = new Dialog();
                        try {
(4)             imagen = Image.createImage(recibido);
(5)             imagen.scale(135, 170);
                        } catch (IOException e) {
                        Dialog.show("IOException", e.getMessage(),
                       "Aceptar", null);
                                }

                        Label label = new Label(imagen);
                        label.setAlignment(Label.CENTER);
                        dialogo.addComponent(label);
                        dialogo.addCommand(new Command("Aceptar"));
                        dialogo.setTitle(subject);
                        dialogo.setDialogType(Dialog.TYPE_ALARM);
                        new Estilos().dialogoEstilo(dialogo);
                        new Estilos().labelEstilo(label);
                        dialogo.show();
}
```

4.3.2. Bluetooth

4.3.2.1. Bluetooth remoto

Esta clase se encarga de realizar la gestión de la comunicación bluetooth, en lo que respecta al funcionamiento como servidor. Debe contener los servicios que ofrecerá a los potenciales clientes, así como un mecanismo de escucha de mensajes. Esta escucha debe estar presente en toda la aplicación, es por este hecho, por el cual el objeto se crea dentro del hilo de ejecución del midlet en el `starApp()`.

Lo primero que debemos hacer es obtener el objeto que gestiona el dispositivo bluetooth del propio terminal (`LocalDevice.getLocalDevice();`). Para después, arrancar el hilo de escucha bluetooth, el cual es una clase interna.

```
            public class HiloBluetooth extends Thread {
            public HiloBluetooth() {
            }
            public void run() {
                  setVisible();
                  conexionServidor();
                  crearServicio();
            }
      }
```

Dentro del método de arranque del hilo se encuentran los métodos necesarios para establecer la visibilidad del dispositivo, las conexiones y crear el servicio:

```
setVisible()
```

Con este método dictaminamos el tipo de visibilidad que tiene el dispositivo al usar esta aplicación, `dispositivoservidor.setDiscoverable(DiscoveryAgent.GIAC);`. DiscoveryAgent es la clase encargada de descubrir los dispositivos y servicios y de dotarles de sus características. Los mensajes de búsqueda no tienen información sobre la fuente emisora, pero si pueden indicar el tipo de dispositivos que deben responder. GIAC (*General Inquiry Acces Code*), es un código de acceso de pregunta que indica el tipo de respuesta que espera del otro dispositivo.

```
      public void setVisible() {
            try {
       dispositivoservidor.setDiscoverable(DiscoveryAgent.GIAC);
            } catch (BluetoothStateException e) {
            Dialog.show("BluetoothStateException",e.getMessage(),
      "Aceptar",null);
            }
       }
```

`conexionServidor()`

Se debe establecer una conexión basada en el perfil del puerto a utilizar, este tipo de conexión se conoce como SSP. Para indicarlo así, debemos utilizar una url determinada para la creación. Además, se debe incluir en dicha url el UUID (*Identificador único universal del servicio*).

Con todo ello, ya se dan las condiciones adecuadas para realizar la conexión, que se realizara mediante un StreamBufferNotifier.

```
public void conexionServidor() {

      StringBuffer url = new StringBuffer("btspp://localhost:");
      url.append((new UUID(0x1234)).toString());
      url.append(";name=Mobile Surveillance;authorize=true");

      try {
            notifier = (StreamConnectionNotifier) Connector
                        .open(url.toString());
      } catch (IOException e) {
            Dialog.show("IOException", e.getMessage(), "Aceptar",
      null);
      }

}
```

`crearServicio()`

La primera acción a realizar es crear el propio servicio, obteniéndolo del propio dispositivo, el servicio estará identificado por el objeto notifier. Para registrar el servicio debemos incluir unos atributos al mismo, un identificador (*ATRIBUTO_MOBILE_SURVEILLANCE*) y los elementos utilizados. En ese caso los elementos serán un objeto enumeration (javax.microedition.enumeration), DATSEQ, con el que indicamos que hay que seleccionar todos los elementos de la lista.

Cuando se produce la conexión con el servicio, hay que indicar que se acepta y se abre la conexión, y posteriormente indicar, cual es la accion a realizar, en este caso finalizar la aplicación.

Para realizar la comprobación de si la palabra enviada es la que posee la aplicación para terminar la ejecución, hay que obtener las dos palabras, y posteriormente compararlas, y en caso afirmativo, llamar al método `destroyApp()`.

La palabra que proviene del mensaje bluetooth, se obtiene de la conexión: `in = conexion.openDataInputStream(); palabraStop=in.readUTF();` mientras que la palabra propia de la aplicación se obtiene de la memoria permanente, recordStore, de forma similar a la obtención de las palabras para la comprobación con los mensajes de texto[ref a donde se indica].

```
public void crearServicio() {
ServiceRecord servicio= dispositivoservidor.getRecord(notifier);
DataElement element = new DataElement(DataElement.DATSEQ);
servicio.setAttributeValue(ATRIBUTO_MOBILE_SURVEILLANCE,element)
      try {
            while (true) {
                  StreamConnection con= notifier.acceptAndOpen();
                  pararAplicacion(obtenerInformacion(con));
            }
      } catch (IOException e) {
            Dialog.show("IOException", e.getMessage(), "Aceptar",
      null);
      }
}
```

4.3.2.2. BluetoothLocal

Esta clase se encarga de realizar la gestión de la comunicación bluetooth, en lo que respecta al funcionamiento como cliente. Para ello implementa la interfaz DiscoveryListener, que provee los métodos para describir dispositivos los servicios disponibles.

El objeto es creado cuando se va a utilizar, es decir, en el menú receptor en el momento que se selecciona la opción de terminar aplicación vía bluetooth. En su constructor, obtiene el objeto que controla la gestión de la comunicación bluetooth del dispositivo (`dispositivolocal =LocalDevice.getLocalDevice();`), indica la visibilidad y obtiene el objeto encargado de encontrar los servicios y dispositivos, que estará asociado al objeto de control(`discoveryagent = dispositivolocal.getDiscoveryAgent();`).

En el momento que se le indica comienza la búsqueda de dispositivos, `cliente.buscarDispositivosRemotos();`. Dentro de este método comienza la búsqueda de dispositivos mediante el inicio de la "encuesta" `discoveryagent.startInquiry(DiscoveryAgent.GIAC, this);`.

Cuando descubre un nuevo dispositivo, se llama al método deviceDiscovered cuya misión será buscar si existe el servicio que necesita la aplicación (1), dentro de los servicios que posee el dispositivo. Cuando descubra el servicio indicado `(SERVICE = MOBILE_SURVEILLANCE, ATRIBUTOS = ATRIBUTO_MOBILE_SURVEILLANCE);` se llamará al método `servicesDiscovered()`, donde se creará la conexión del cliente, en la cual se introduce como dato de salida (4) la palabra de fin (obtenida del sistema RMS). La conexión del cliente (3) con el servicio se hace en función de la url obtenida del propio servicio (2).

```
public void deviceDiscovered(RemoteDevice dispositivoremoto,
            DeviceClass tipodispositivo) {

        try {
        (1)  discoveryagent.searchServices(ATRIBUTOS, SERVICIOS,
         dispositivoremoto, this);

        } catch (BluetoothStateException e) {
        Dialog.show("MediaException", e.getMessage(),
        "Aceptar", null);
        }
}
```

```
	public void servicesDiscovered(int transID, ServiceRecord[]
	servRecord) {
		ServiceRecord servicio = null;
		for (int i = 0; i < servRecord.length; i++)
			servicio = servRecord[i];
			conexionCliente(servicio);
		}
	}

	public void conexionCliente(ServiceRecord serv) {
		(2)	String url = serv.getConnectionURL(
				ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
		StreamConnection connection = null;
		DataInputStream in = null;
		DataOutputStream out = null;
		try {
		(3)	connection = (StreamConnection) Connector.open(url);
			in = connection.openDataInputStream();
		(4)	out = connection.openDataOutputStream();
			out.writeUTF(obtenerpalabraFinLocal());
			out.flush();
		} catch (IOException e) {
			Dialog.show("IOException", e.getMessage(), "Aceptar",
			null);
		} finally {
			try {
				if (connection != null) {
					connection.close();}
				if (in != null) {
					in.close(); }
				if (out != null) {
					out.close();}
			} catch (IOException e) {
				Dialog.show("IOException", e.getMessage(),
				"Aceptar",null);
			}
		}
	}
```

4.4. Estilos

A través de esta clase, y las llamadas que se realizan a ella a lo largo de la aplicación se controla y genera la apariencia de todos los componentes.

```
public Estilos()
```

No recibe ningún parámetro de entrada y en su constructor no realiza ninguna acción, puesto que al crear un objeto de esta clase, lo que nos interesa es acceder a sus métodos.

Estos métodos se encargaran de cambiar o modificar diferentes aspectos visuales de los componentes sobre los que actúa.

Sobre un formulario se establece el color de fondo (1) y la apariencia del título (2) y la barra de menú (3).

```
public void formularioEstilo(Form formulario) {
      estiloFormulario = new Style();
(1)   estiloFormulario.setBgColor(0xF84803);
(2)   formulario.setTitleStyle(tituloEstilo());
(3)   formulario.setSoftButtonStyle
      (barraMenuEstilo());
      formulario.setStyle(estiloFormulario);
}
```

Sobre un botón se indica el alineamiento (1) y situamos la imagen de fondo (2).

```
public void botonEstilo(Button boton) {
      estiloBoton = new Style();
(1)   boton.setAlignment(Button.CENTER);
(2)   estiloBoton.setBgImage
      (Image.createImage("/botonMenu.jpg"));
      boton.setStyle(estiloBoton);
}
public void botonPequeñoEstilo(Button boton) {
      estiloBoton = new Style();
      boton.setAlignment(Button.CENTER);
```

```
        estiloBoton.setBgImage
        (Image.createImage("/botonMenuPequeno.jpg"));
        boton.setStyle(estiloBoton);
    }
```

Sobre un contenedor de pestañas se indica el color de fondo, y los colores de selección (1).

```
    Public void pestañaEstilo(TabbedPane pestaña) {
        estiloPestaña = new Style();
        estiloPestaña.setBgColor(0xf5b658);
(1)     estiloPestaña.setBgSelectionColor(0x3300ff);
        estiloPestaña.setFgSelectionColor(0x3300ff);
        estiloPestaña.setFgColor(0x3300ff);
        pestaña.setStyle(estiloPestaña);
    }
```

Sobre un dialogo, se establece la apariencia del título y la barra de menú.

```
    public void dialogoEstilo(Dialog dialogo) {
        dialogo.setTitleStyle(tituloEstilo());
        dialogo.setSoftButtonStyle(barraMenuEstilo());
    }
```

Sobre un cuadro de texto se establece el color de selección.

```
    public void textFieldEstilo(TextField texto) {
        estilotextField = new Style();
        estilotextField.setBgSelectionColor(0x3300ff);
        texto.setStyle(estilotextField);
    }
```

Sobre un área de texto se indica el color de fondo y los de selección.

```
    public void textAreaEstilo(TextArea texto) {
        estiloTextArea = new Style();
        estiloTextArea.setBgColor(0xf5b658);
        estiloTextArea.setBgSelectionColor(0xf5b658);
        estiloTextArea.setFgSelectionColor(0);
        texto.setStyle(estiloTextArea);
    }
```

Se crea el estilo de la barra de menú modificando el color de fondo.

```
        public Style barraMenuEstilo() {
        estiloBarraMenu = new Style();
        estiloBarraMenu.setBgColor(0xff8800);
        return estiloBarraMenu;
    }
```

Se crea el estilo del título modificando el color de fondo.

```
public Style tituloEstilo() {
      estiloTitulo = new Style();
      estiloTitulo.setBgColor(0xff8800);
      return estiloTitulo;
}
```

Sobre un radioButton se establece el color de fondo y el de selección.

```
public void radioButtonEstilo(RadioButton boton) {
     estiloRadioButton = new Style();
     estiloRadioButton.setBgColor(0xff8800);
     estiloRadioButton.setBgSelectionColor(0x3300ff);
     boton.setStyle(estiloRadioButton);
}
```

Sobre un contenedor se establece el color de fondo.

```
public void contenedorEstilo(Container contenedor) {
      estiloContenedor = new Style();
      estiloContenedor.setBgColor(0xf5b658);
      contenedor.setStyle(estiloContenedor);
}
```

Sobre un Label se establece el alineamiento, el color de fondo y se activa la función de ticker [5](1)

```
public void labelEstilo(Label label) {
      estiloLabel = new Style();
      label.setAlignment(Label.CENTER);
(1)   label.setTickerEnabled(true);
      estiloLabel.setBgColor(0xf5b658);
      label.setStyle(estiloLabel);
}
```

[5] *Desplazamiento horizontal del texto cuando su longitud es mayor que el ancho de la pantalla.*

5. Posibles mejoras y trabajos futuros

A la hora de realizar el análisis de los elementos que se pueden mejorar de esta aplicación y de los trabajos futuros, es conveniente establecer las barreras tecnológicas existentes, para poder establecer los objetivos de manera realista, sin dejarnos llevar por la imaginación de una aplicación tan futurista como irrealizable.

Actualmente el principal impedimento está relacionado con la autonomía, no existe ningún terminal que pueda estar encendido indefinidamente, ni siquiera durante un gran número de horas, y mucho menos encendido y realizando acciones, por lo que el hecho de tener que enchufar a la red el terminal siempre será una barrera.

La capacidad de procesamiento de un dispositivo y su capacidad de memoria, aunque están en constante evolución es también limitada, por lo que para mejorar el algoritmo de detección, aunque actualmente es bastante preciso, siempre estará supeditado a la carga computacional. Es una buena idea, poder realizar un pequeño historial de imágenes en las que se ha producido una detección en el propio terminal móvil, pero aunque extensa, la memoria es limitada, por lo que el sistema deberá tener un límite máximo de almacenamiento y un sistema de actualización cuando se acerque a la máxima ocupación. Además, el límite ha de ser dinámico, ya que la memoria la comparten el resto de aplicaciones del teléfono. Este punto, de historial de imágenes, actualmente, siguiendo los requisitos expuestos, si es realizable.

La mayoría de los terminales que disponen de cámara proporcionan la opción de grabar vídeo, en lo que a programación se refiere se utiliza en mismo API (MMAPI), para controlar este recurso. Por tanto, una funcionalidad no implementada que sería interesante, es la opción de en vez de tomar fotos, tomar vídeos a partir del momento en que se ha detectado el movimiento. En cuanto al envío, se puede proceder de la misma manera que actualmente se realiza con una sola imagen. Siempre, teniendo en cuenta,

que un vídeo, evidentemente, ocupa una mayor cantidad de memoria y su tratamiento es mucho más costoso, por lo que el tiempo de grabación debería estar muy limitado, a unos pocos segundos.

Se pueden aumentar las opciones de manejo remoto a través de otro terminal: petición de grabación de vídeo, petición de toma de una foto, vista del historial. Además, este manejo remoto, también se podría implementar a través del servidor Web, esto supondría el uso de un servidor de envío de mensajes propio o utilizar un servicio de terceros, y tendría un coste asociado. Para el envío de la información al servidor desde el terminal, se puede utilizar la conexión a Internet del propio terminal, teniendo en cuenta también su coste, pero es una funcionalidad que se puede añadir actualmente a este proyecto. Hay que destacar, que esta funcionalidad, actualmente ha de ser mediante un servicio de petición, espera a que se generen todos los datos, envío y visualización, nunca en tiempo real. La implementación de un servicio "streaming" con un teléfono como fuente es muy complicada, de hecho el propio API, no da la opción de realizarlo. Pero en el futuro seguro que será un servicio integrado en todos los terminales.

LWUIT es un motor grafico en constante evolución, este proyecto se ha realizado con una versión anterior a la que existe en la actualidad [ver apartado 2.2.4.6], y cada nueva versión, corrige los problemas de la anterior (genera nuevos), y permite más opciones. Si se realizan varias versiones con el tiempo, sería recomendable utilizar la última versión existente en cada momento.

Esta aplicación es compatible con un gran rango de terminales, en función de sus prestaciones. Evidentemente, es imposible realizar las pruebas en todos los terminales del mundo, y tampoco hemos dispuesto de una gran variedad de ellos, la gran mayoría de pruebas se han realizado en un Nokia 5800 Xpress Music de manera satisfactoria, y en Nokia 6131, con buenos resultados. Para una futura implantación serían necesarias más pruebas, que se escapan de la logística de este proyecto, aunque es especialmente interesante el uso de la herramienta deviceAnyWhere [34], que permite la prueba real en terminales de forma remota. Un trabajo futuro para la implementación sería, sin duda, una fase de prueba y verificación en muchos terminales.

6. Conclusiones

El principal objetivo de este proyecto era realizar un sistema de televigilancia en tiempo real utilizando como elemento de captura la cámara de un terminal móvil, con capacidad de dar aviso a otro terminal. Por lo que ese objetivo principal, ha sido llevado a cabo con éxito. Además se le ha dotado del apoyo de un servidor, y de la posibilidad de manejo de forma remota por parte de otro terminal, lo que supone un proyecto de mayor envergadura, mucho más completo y atractivo para el fin último de cualquier aplicación, su implantación en el mundo real.

Según se puede comprobar en el estado del arte, no se trata de una aplicación novedosa en cuanto a concepto, aunque su idea es de sistema de televigilancia, no solo de sistema espía (de dudosa legalidad). Pero el resto de aplicaciones, quedan descompensadas por falta de funcionalidad y por una gran complejidad en su uso, además de en el caso de Digia ImageSpy, una casi imposible ejecución en terminales reales. Con esta aplicación, se ha conseguido un compendio de las aplicaciones existentes, con una mejora sustancial en el apartado visual, sencillez de manejo, y teóricamente válida para los terminales con capacidades suficientes.

Nunca se trata de sustituir a un sistema de televigilancia clásico, esto serían sin duda, una utopía. En un primer caso, porque, aunque mejore mucho la tecnología, la cámara del teléfono no está pensada ni diseñada para la vigilancia, y nunca se podrá comparar con una cámara especialmente pensada y diseñada para ello. Además que al utilizar esta aplicación, se inutiliza la función básica de cualquier teléfono, que no es otra que realizar llamadas.

Esta aplicación, no evita que se produzca el delito (¿Qué sistema de vigilancia es capaz de evitarlo?), pero es rápida en su detección, además de silenciosa, lo que permite que se detecte la situación anómala sin que el infractor se de cuenta que es observado y proporciona una mayor probabilidad de actuación externa mientras tras la detección, en un menor espacio de tiempo.

Esta aplicación, no sería recomendable para proyectos de vigilancia de gran envergadura, de grandes edificios, donde el rigor y la calidad de imagen es vital. Pero para un sistema sencillo, para el hogar, supone una aplicación económica y segura, que no tiene nada que envidiar a cualquier solución para el hogar. No necesita mantenimiento especializado, ni instalaciones, ni siquiera una tarifa especial por usar el servicio, solo necesitas tu teléfono móvil, elemento que en la actualidad, en la sociedad desarrollada, se considera indispensable.

Por tanto, el aspecto económico también es un punto a favor, si se da la situación ideal, ninguna situación provoca la detección, el coste asociado es el precio de la aplicación, y del teléfono (normalmente no se compra solo para esta funcionalidad), el cual es mucho más barato que cualquier sistema de televigilancia. Y en el peor de los casos, el coste aumentará en el precio de un mensaje multimedia. Ya que la configuración y el manejo remoto, si se desea se puede realizar a coste cero.

7. Anexo I. Planificación

7.1. Tareas.

Para realizar la planificación de este proyecto se ha decidido dividir dicho proyecto en las siguientes tareas:

Nombre de la tarea	Análisis de requisitos
Descripción	Análisis de requisitos, funcionalidad y tecnología a utilizar en el dispositivo móvil
Duración	15 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Formación en J2ME (J2ME, LWUIT, MMAPI)
Descripción	Adquisición de conocimientos previos para la implementación de la funcionalidad básica y la interfaz gráfica.
Duración	15 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Diseño e implementación de la interfaz gráfica
Descripción	Diseño de la interfaz gráfica de la aplicación.
Duración	10 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Implementación del sistema de detección de movimiento
Descripción	Implementación del algoritmo de detección de movimiento
Duración	10 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Formación WMA
Descripción	Adquisición de conocimientos previos para la implementación de la comunicación vía mensajes
Duración	5 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Implementación de las comunicaciones para el sistema móvil
Descripción	Implementación de las comunicaciones vía mensajes
Duración	5 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Formación Bluetooth
Descripción	Adquisición de conocimientos previos para la implementación de la comunicación Bluetooth
Duración	3 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Implementación comunicación Bluetooth
Descripción	Implementación de la comunicación bluetooth entre los terminales
Duración	4 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Pruebas
Descripción	Pruebas de integración, unitarias sobre emulador, corrección de errores y prueba sobre terminal móvil
Duración	10 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones; Móvil Nokia 6131[1];Móvil Nokia 5800[1]; Presupuesto mensajería

Nombre de la tarea	Pruebas Sistema completo
Descripción	Pruebas de integración de ambos proyectos y corrección de errores
Duración	4 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones [2]; Servidor Web Apache Tomcat[1];Servidor Web Apache con soporte para SQL[1];Móvil Nokia 6131[1];Móvil Nokia 5800[1]: Presupuesto mensajería

Nombre de la tarea	Redacción de la memoria del proyecto
Descripción	Redacción de la memoria técnica del proyecto.
Duración	60 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

7.2. Diagrama Gantt

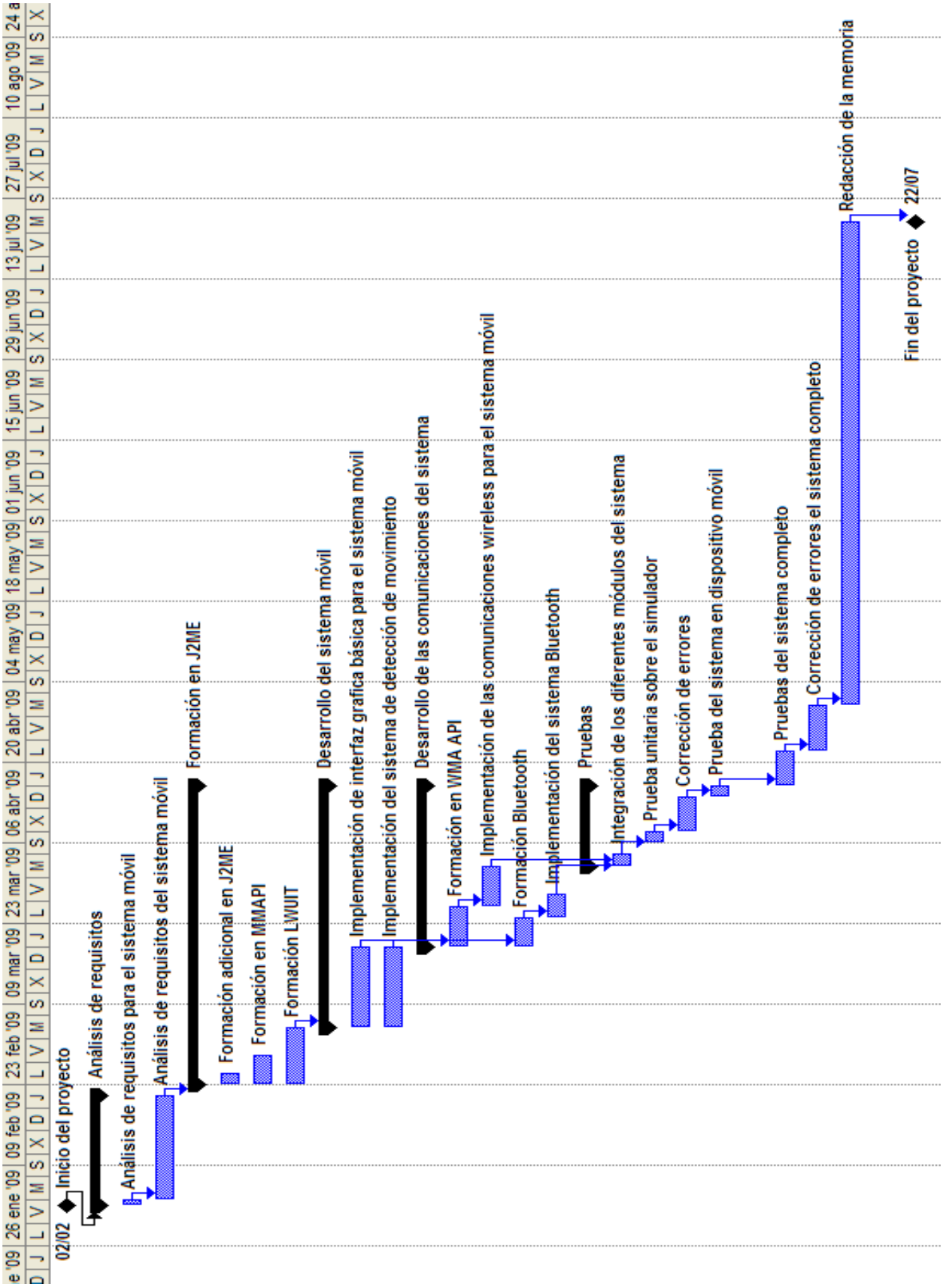

Figura 51: Diagrama Gantt

7.3. Presupuesto

Duración total del proyecto			
Días		114 días	
Horas		912 horas	
Recursos Utilizados			
Recurso	Ingeniero Técnico de Telecomunicaciones	Coste	30 euros/hora
Recurso	Ordenador personal Toshiba	Coste	800 euros
Recurso	Teléfono Nokia N6800	Coste	279 euros
Recurso	Teléfono Nokia 6131	Coste	150 euros
Recurso	Presupuesto Mensajería	Coste	30 euros/uso

Coste Total del proyecto		
Coste asociado a la mano de obra	Coste por horas	30 euros
	Número de horas de trabajo	912 horas
	Coste total	27.360 euros
Coste asociado a los materiales utilizados	1.259 euros	
Coste total del proyecto	28.619 euros	

7.4. Presupuesto global

Tareas	Duración	Coste
Desarrollo de un sistema de tele-vigilancia a través de un dispositivo móvil	912 horas	28.619 euros
Integración de un sistema de tele-vigilancia a través de un dispositivo móvil en un servidor Web	848 horas	26.120 euros
Duración Total		
1396 horas (242 días)		
Presupuesto total		
54.739 euros		

8. Anexo II. Diagramas de flujo

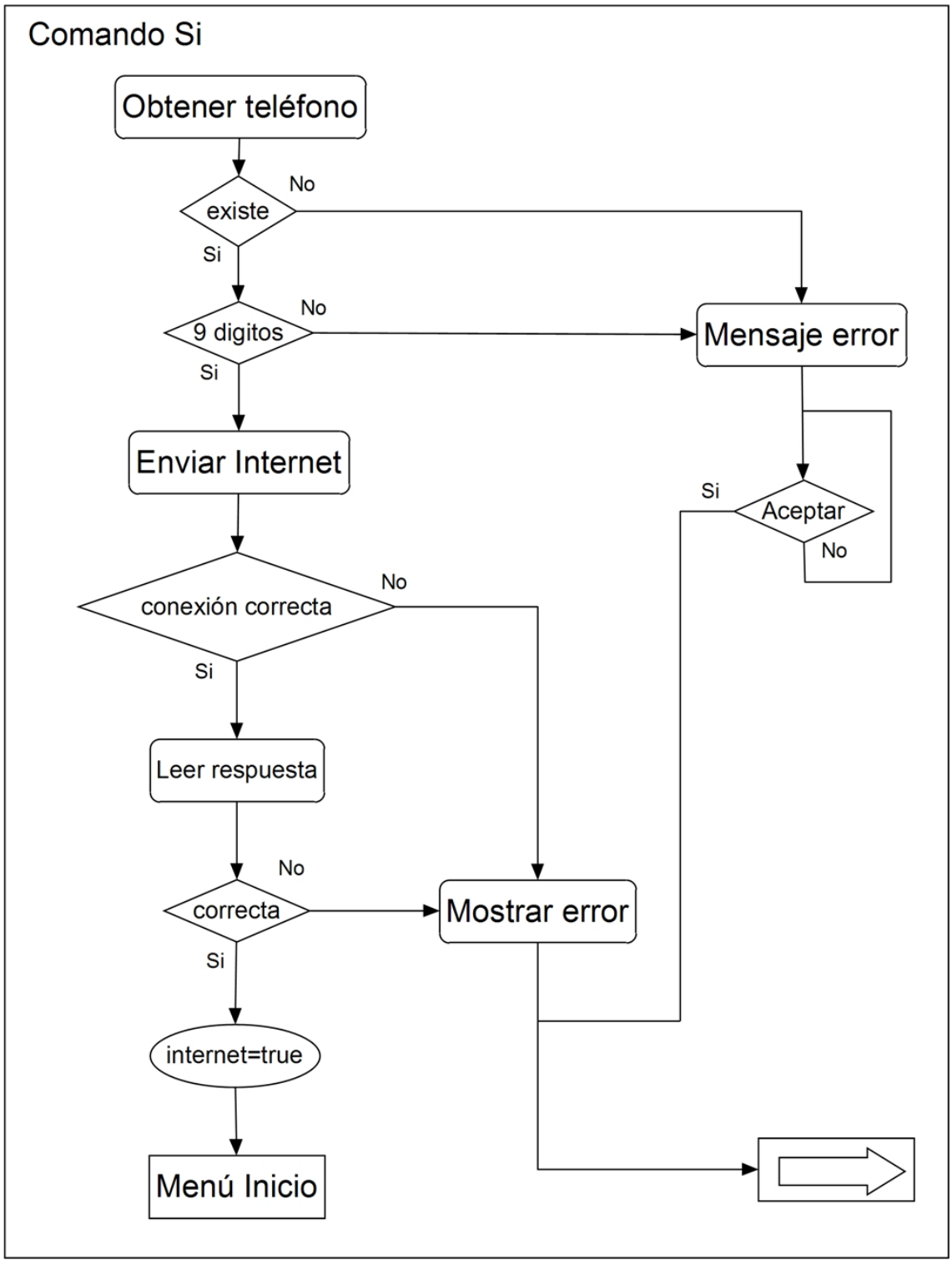

Figura 52: Diagrama de flujo, comando Si.

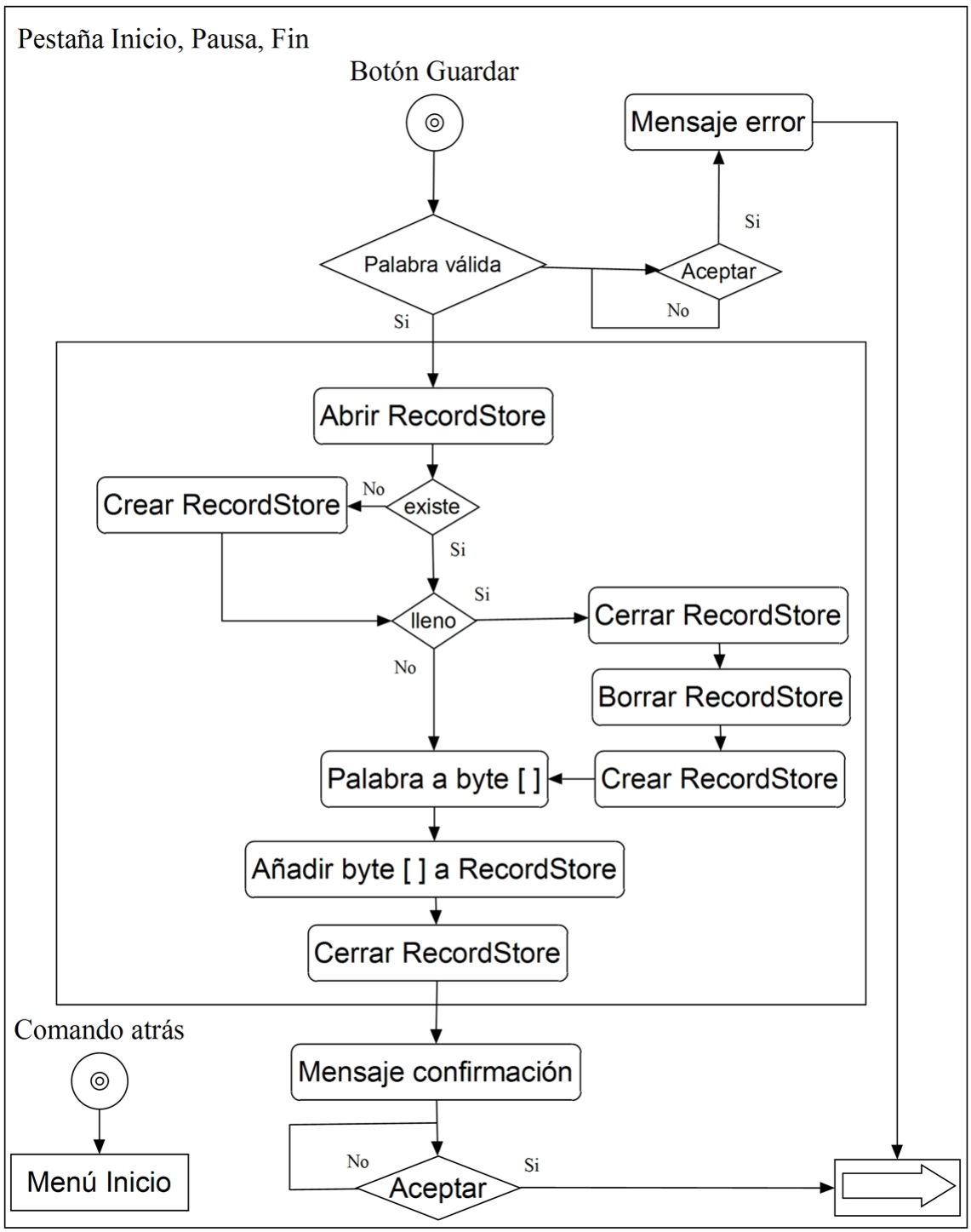

Figura 53: Diagrama de flujo, pestaña inicio, pausa y fin, aplicación captura

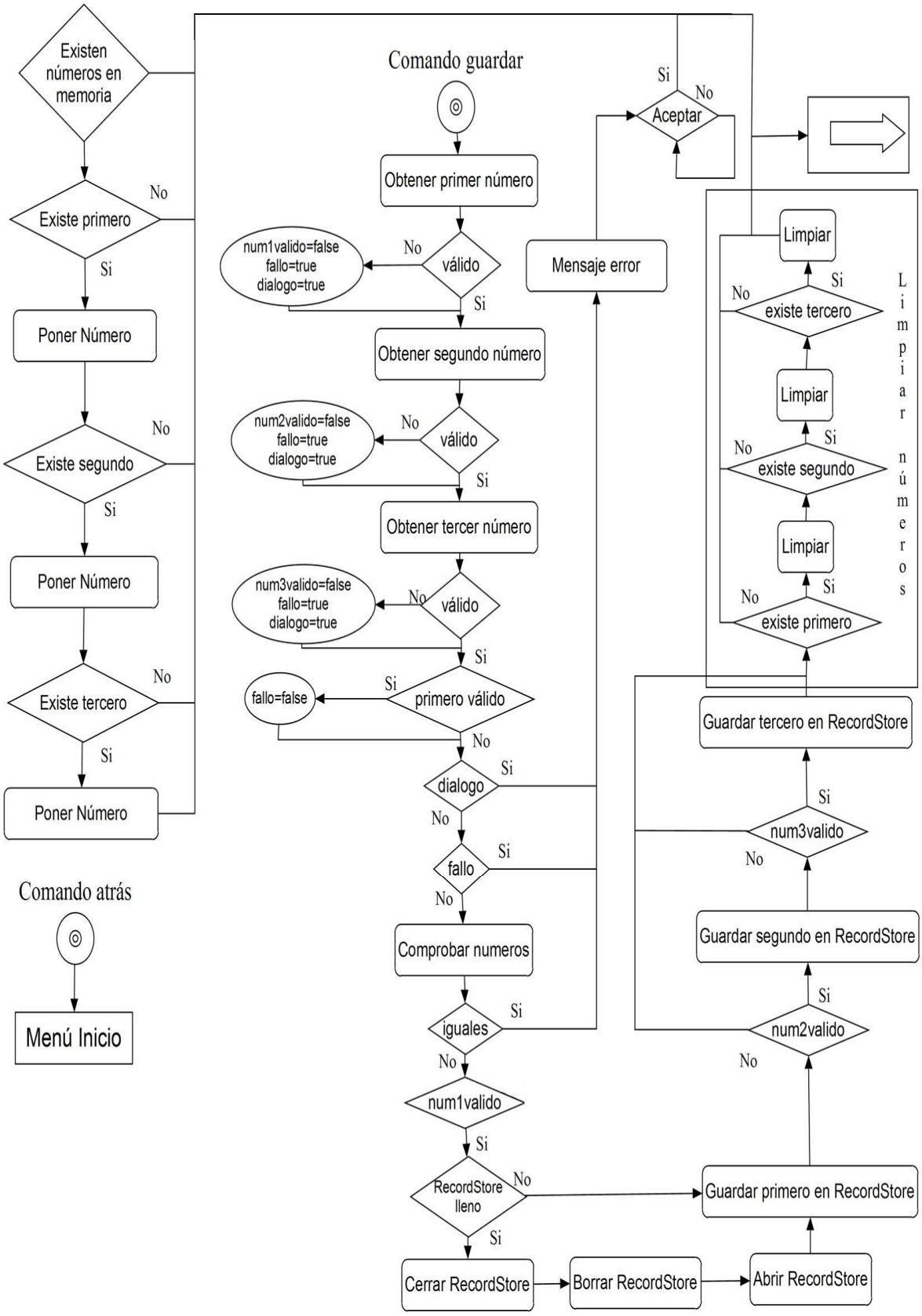

Figura 54: Diagrama de flujo, pestaña números, aplicación captura

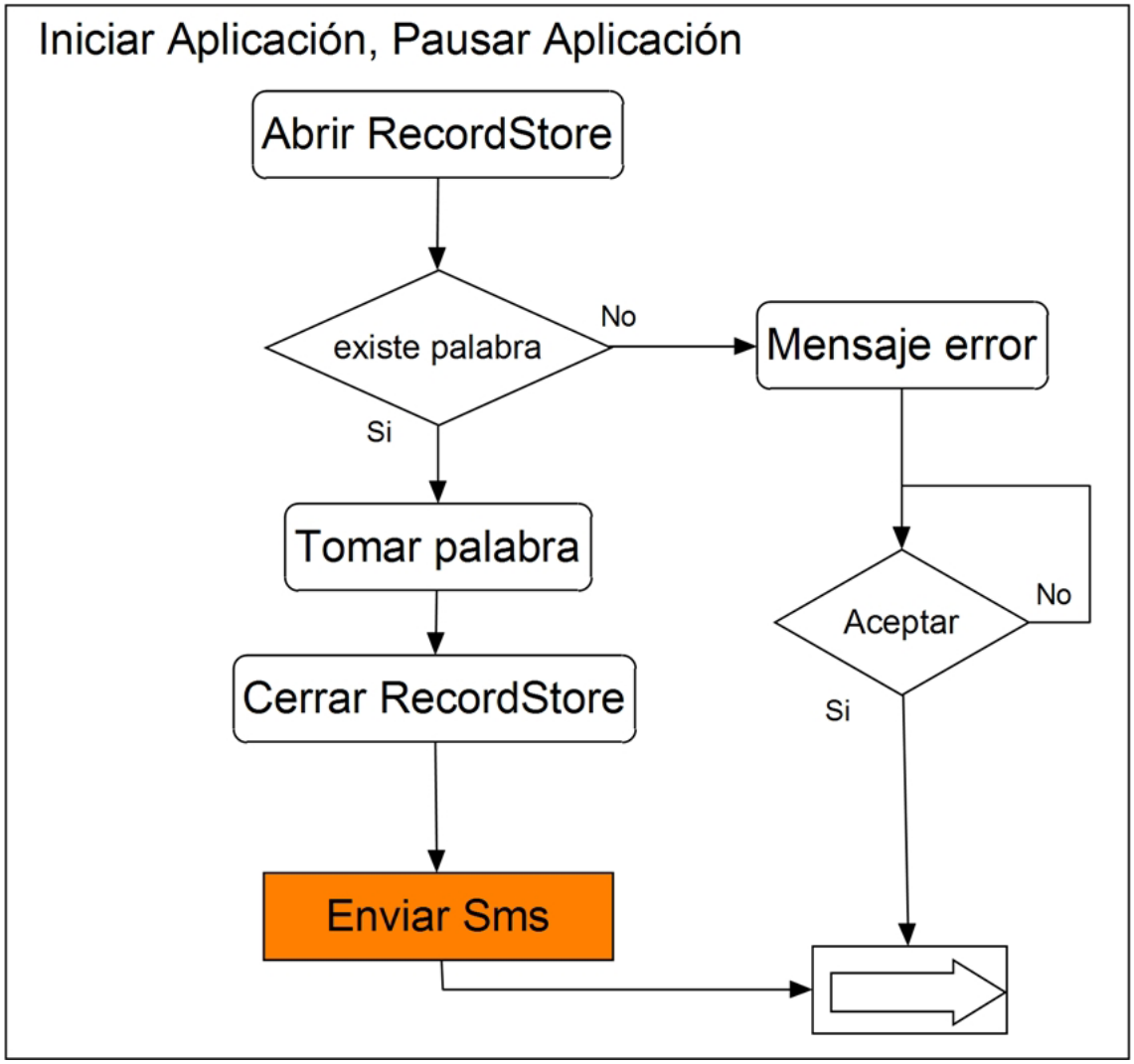

Figura 55: Diagrama de flujo, botones iniciar y pausar aplicación

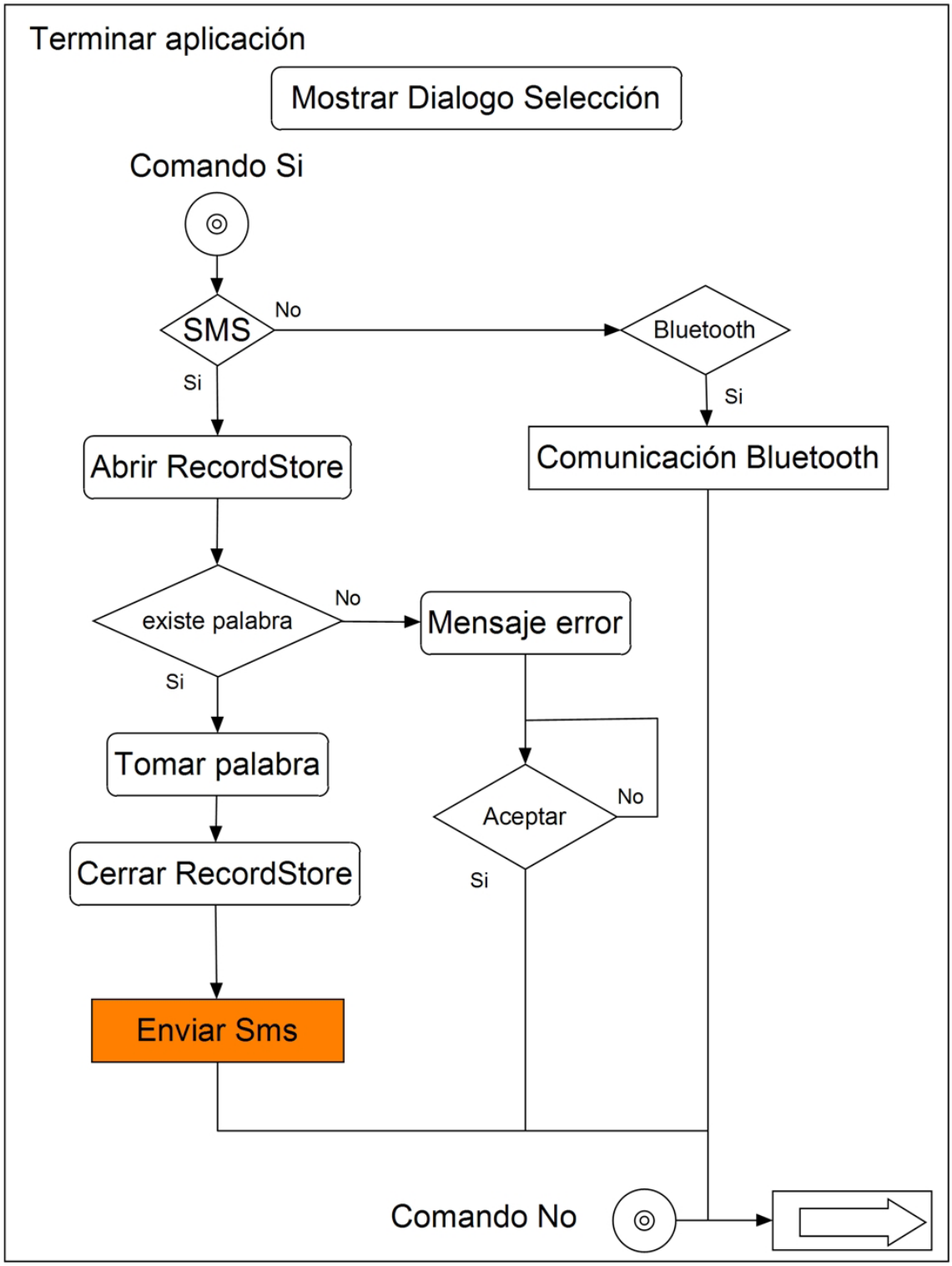

Figura 56: Diagrama de flujo, botón terminar aplicación

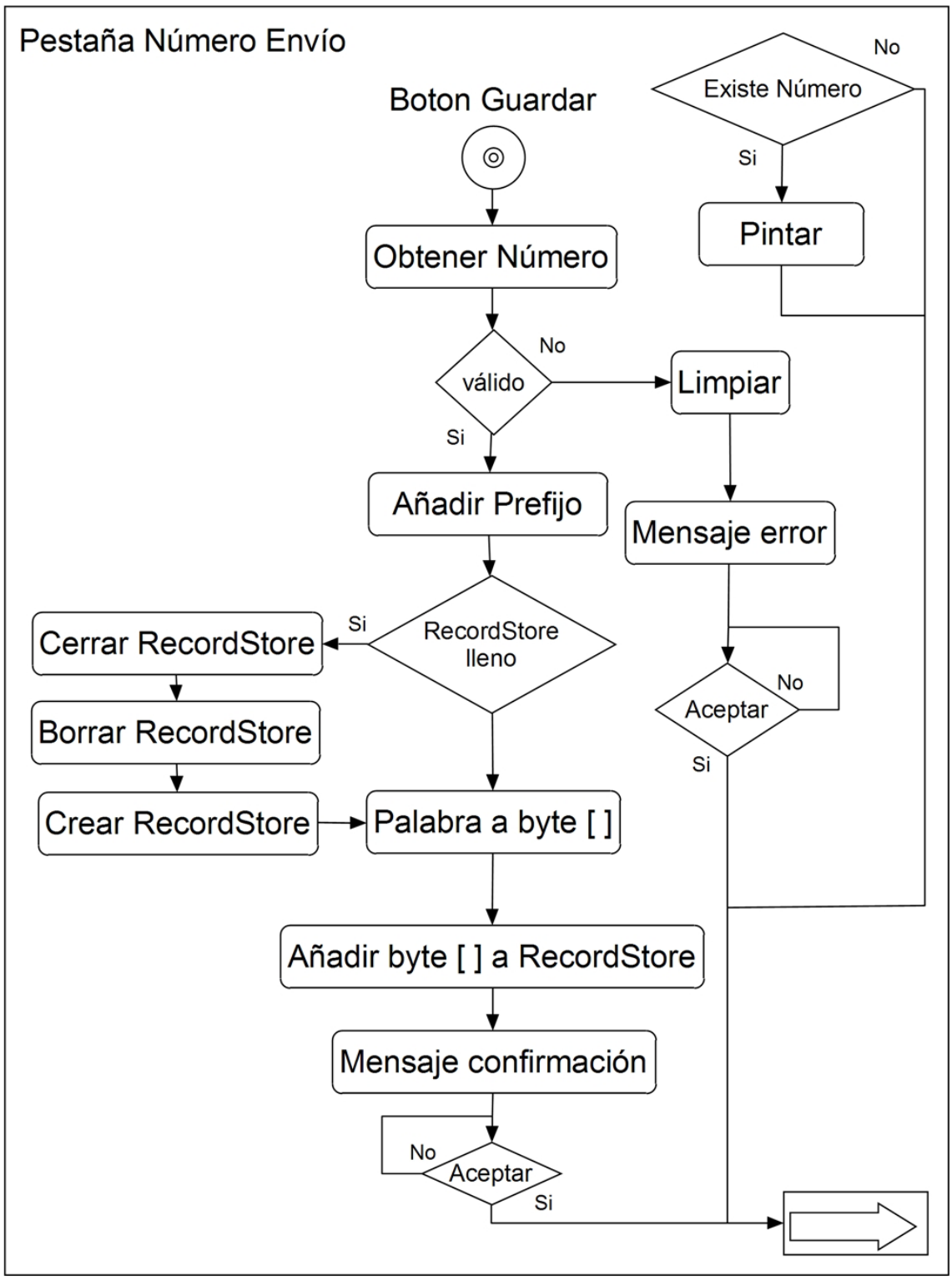

Figura 57: Diagrama de flujo, pestaña número envío, aplicación receptor

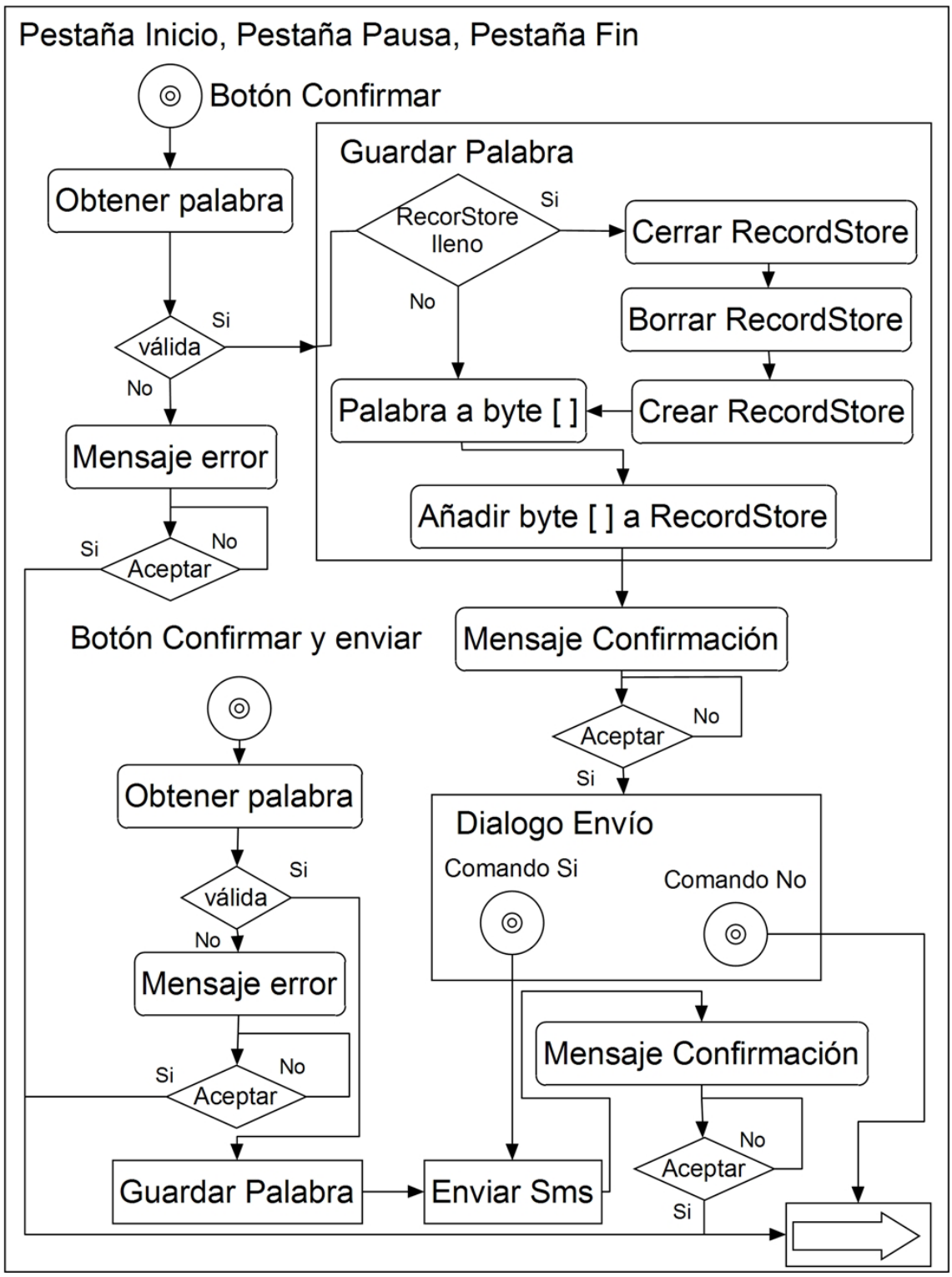

Figura 58: Diagrama de flujo, pestaña inicio, pausa y fin, aplicación receptor

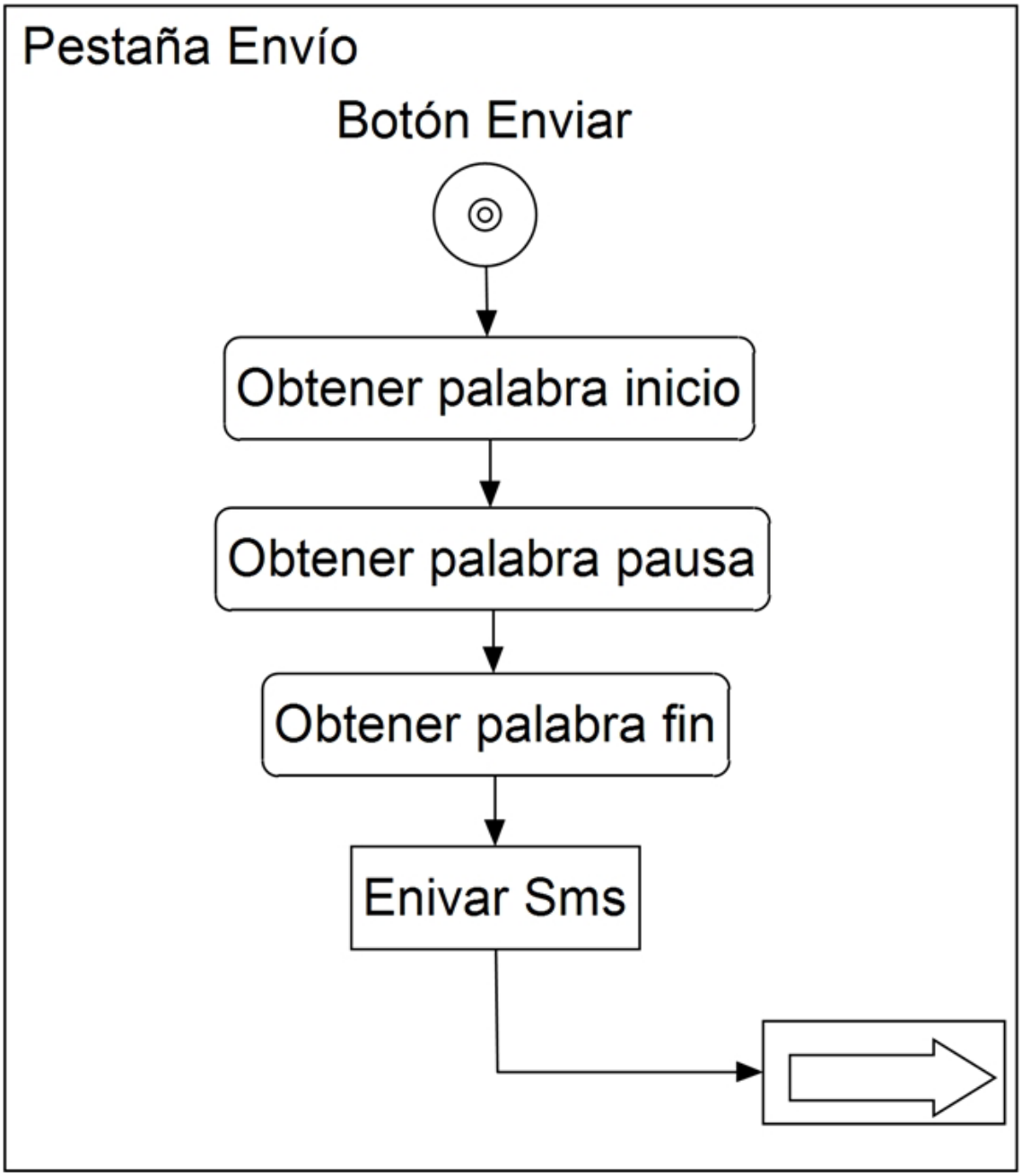

Figura 59: Diagrama de flujo, pestaña envío, aplicación receptor

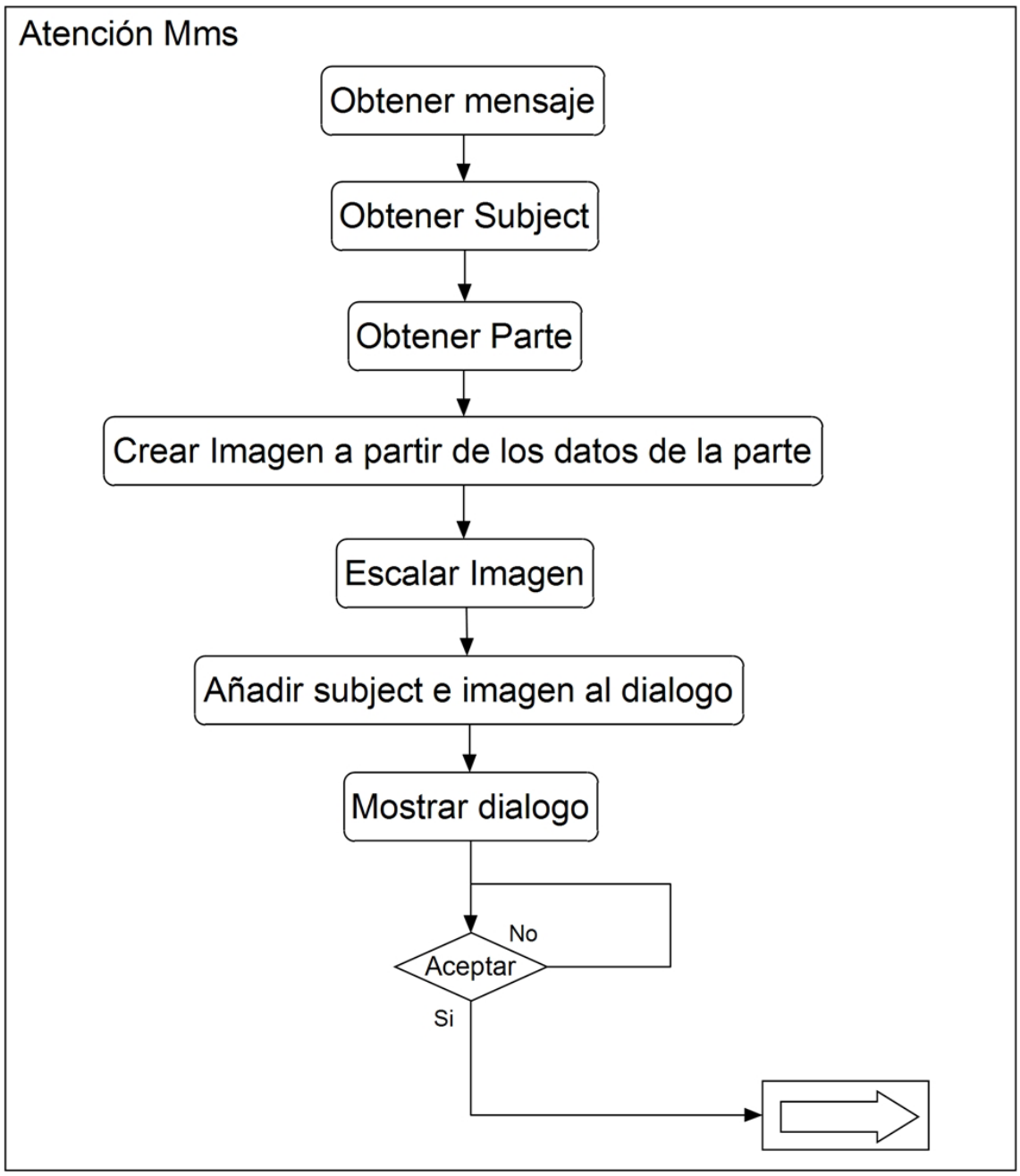

Figura 60: Diagrama de flujo, atención MMS

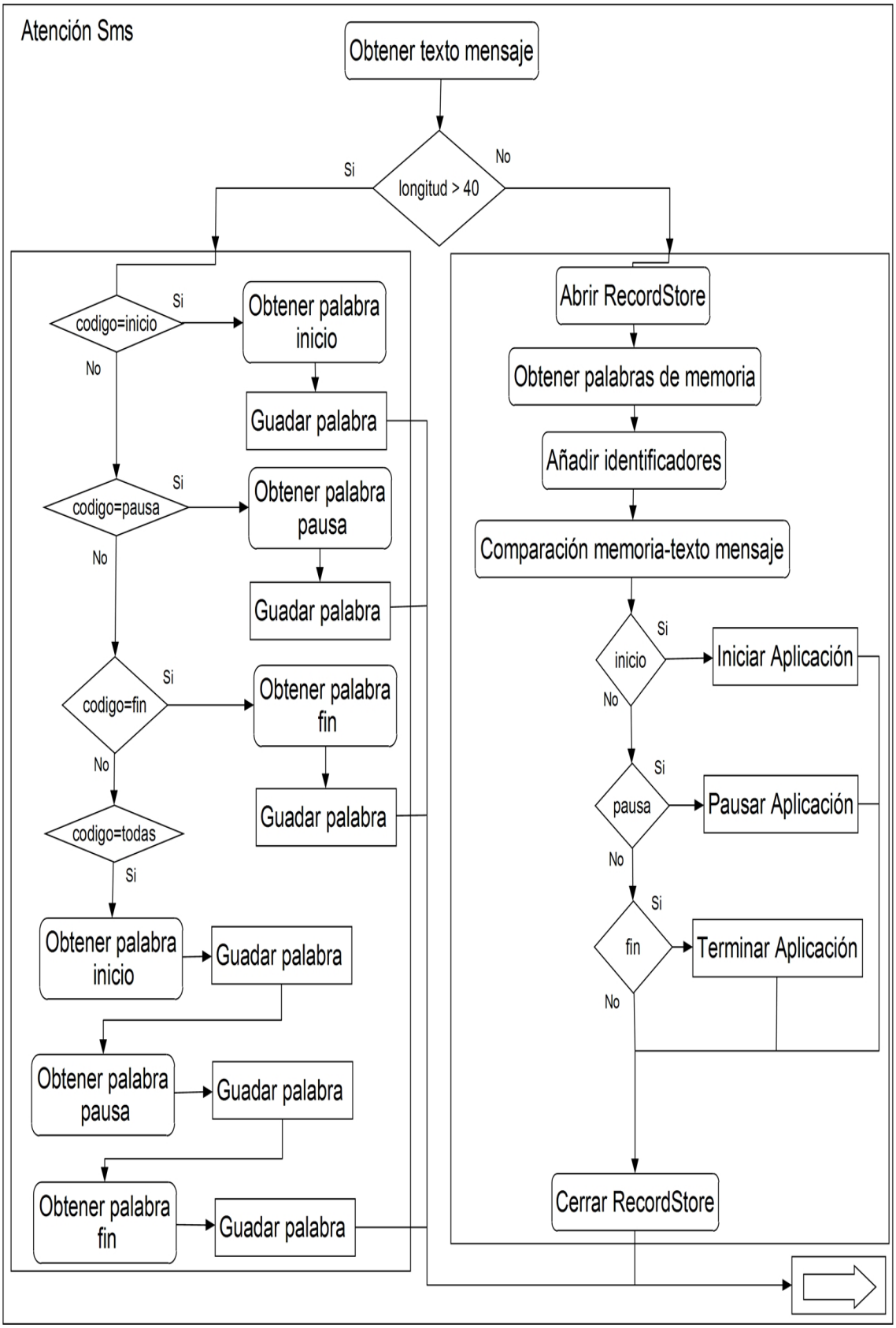

Figura 61: Diagrama de flujo, atención SMS

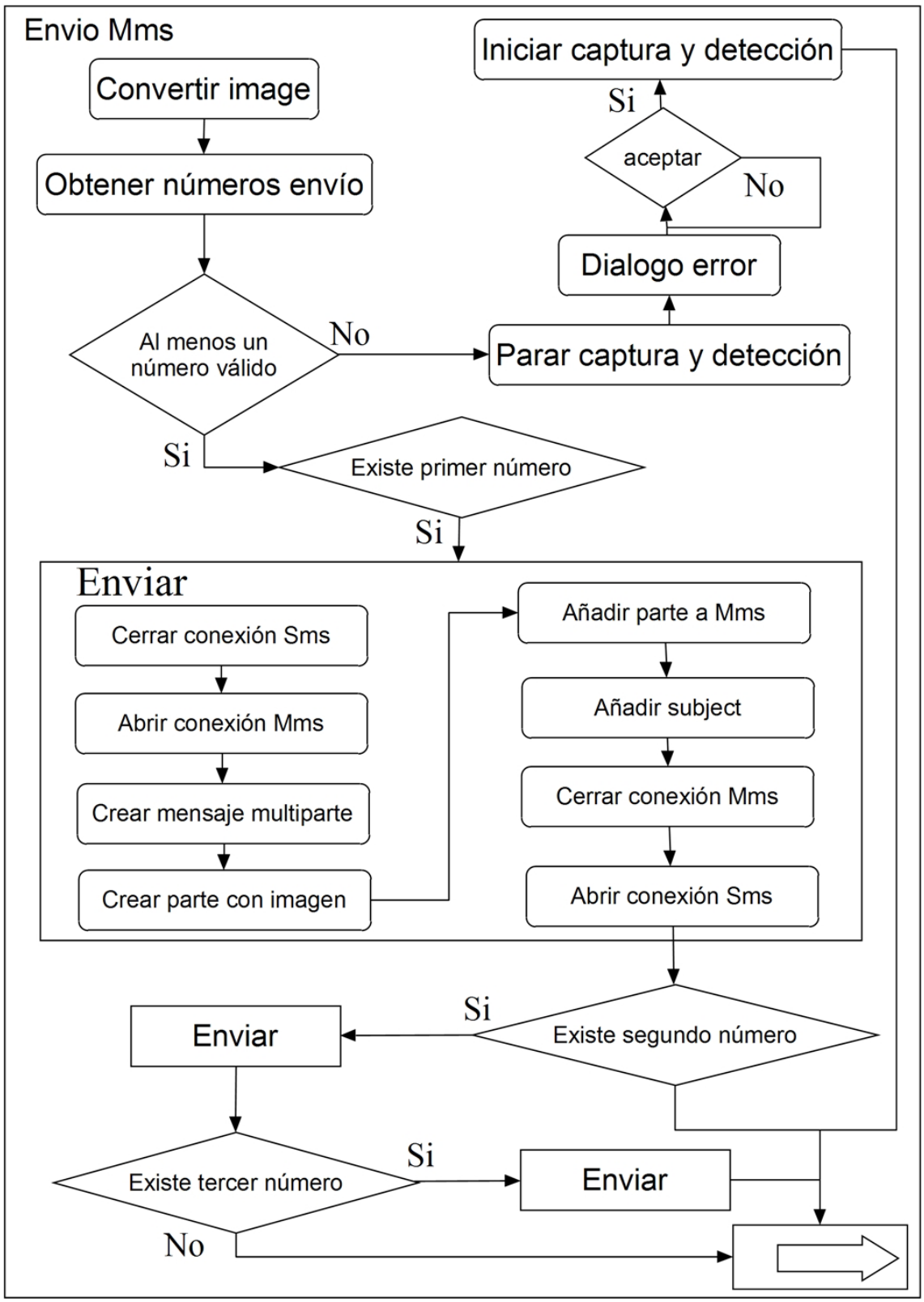

Figura 62: Diagrama de flujo, envío MMS

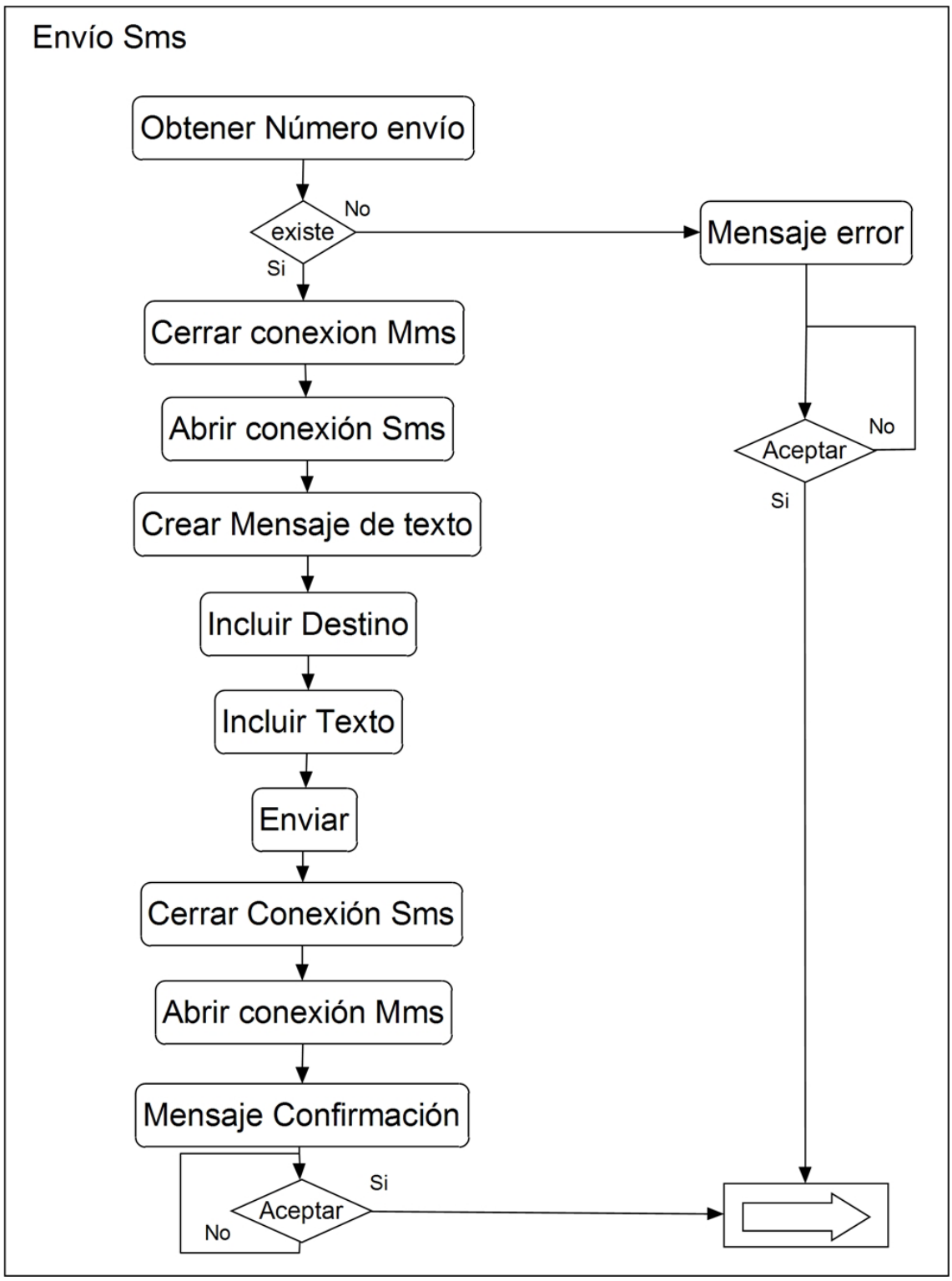

Figura 63: Diagrama de flujo, envío SMS

9. Anexo III. Guía de uso

Dentro de este anexo se sitúa el manual de instrucciones para un posible usuario final, así como para la prueba en el emulador de la aplicación.

9.1. Antes de instalar

En el paquete aparecen dos aplicaciones para instalar:

PFC: En él se encuentra la aplicación de vídeo vigilancia en dos versiones.

Características del dispositivo: Con esta aplicación se obtienen las características técnicas del teléfono. En caso de que la aplicación principal no funcione correctamente es recomendable utilizar esta aplicación para evitar que existan incompatibilidades con el terminal.

Se debe atender a la versión Midp utilizada por el dispositivo, así como la capacidad de captura de vídeo que tiene.

La utilización de la versión adecuada se incluye en la tabla adjunta:

	Mid 2.0	Mid 2.1	Capture://image	Capture://video
Gama alta		X		X
Gama media	X			X
Gama baja	X		X	

Tabla 5: Correspondencia gama-características

Se recomienda antes de la instalación usar en todo caso la aplicación de características, para asegurar que se instala la versión adecuada.

9.2. Instalación

Para la instalación introduzca la aplicación en su terminal (cable, infrarrojo, bluetooth…). Y siga los pasos que en él se le indican para una correcta instalación.

En caso de que se cancele la instalación con el mensaje de error "fichero .jar no valido", está utilizando una versión incompatible con su teléfono. En este caso se recomienda utilizar la aplicación de características para elegir la versión adecuada [33].

9.3. Antes de empezar

Para el correcto funcionamiento de la aplicación, y para evitar posibles interrupciones durante su ejecución, se recomienda realizar los siguientes cambios en la configuración del teléfono destinado a la captura de imágenes (modo captura).

- Restringir todo tipo de llamadas. La llamada es una acción que bloquea el resto de aplicaciones del teléfono y puede no recuperarse la ejecución de manera correcta.

- Permitir a la aplicación el acceso a los recursos multimedia del teléfono (cámara)

- Permitir a la aplicación el envío de mensajería.

- Activar el bluetooth del terminal para la posible comunicación con dicho sistema.

- Permitir a la aplicación la conexión vía bluetooth.

- En caso de querer utilizar la conexión vía Internet, abrir dicha conexión, ya sea WIFI o por canales de pago propios, y permitir a la aplicación el acceso a dicha conexión.

Si usted dispone de una versión firmada, los accesos a los recursos del teléfono no deberían ser necesarios, pero aun así, para evitar fallos por la ausencia de permisos sería muy recomendable.

En el modo receptor, no es necesario (aunque si recomendable para evitar incomodidades) dar permiso a la aplicación. NO se deben restringir las llamadas.

El modo receptor se utiliza para la configuración y utilización de la aplicación captura de manera remota, por lo que no será necesario que el terminal receptor tenga activa la aplicación para la recepción del posible mensaje Multimedia.

9.4. Captura

9.4.1. Menú Inicio

Una vez se arranca la aplicación aparece una primera pantalla encargada de la gestión del acceso a Internet.

Con el comando "Si" se acepta el uso de la conexión a Internet.

En el cuadro de texto deberá introducir el número de teléfono propio del terminal que está usando.

Para un uso correcto, previamente debe estar registrado en el portal web, en caso contrario no será posible la conexión a Internet.

Si se produce algún tipo de error (fallo de conexión, servidor, registro…) aparecerá un mensaje informativo

Con el comando "No" se indica que no se quiere utilizar la conexión a Internet y la funcionalidad asociada a la misma.

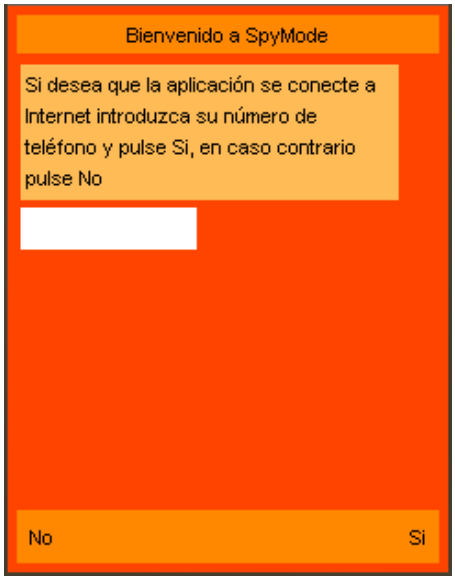

Figura 64: Pantalla Inicio

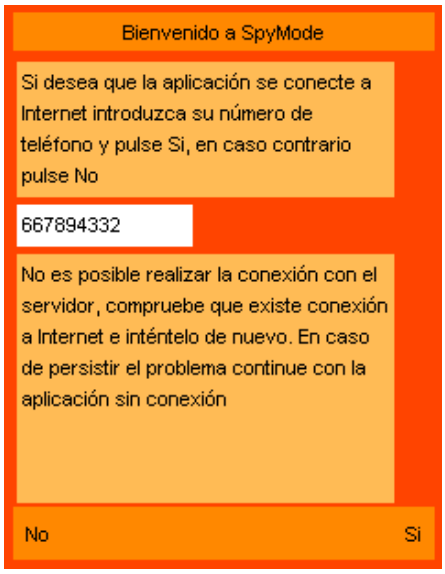

Figura 65: Pantalla Inicio sin conexión a servidor

Esta aplicación es la encargada de la captura de imágenes mediante la cámara del teléfono, de realizar el tratamiento de dichas imágenes y en caso de detectar movimiento, enviará el mensaje multimedia al terminal receptor con la imagen en la que se ha detectado el movimiento. El envío del MMS se realizará siempre, a su vez, si se ha aceptado la conexión a Internet, la imagen se subirá al servidor, este la guardará en la base de datos del usuario y enviara en un e-mail dicha imagen.

9.4.2. Menú Captura

Dentro de este menú podrá seleccionar si desea arrancar la aplicación o configurarla.

Si es la primera vez que arranca la aplicación en su terminal, es totalmente necesario que realice la configuración.

Al pulsar el botón de inicio, se producirá el acceso a la cámara y empezara a correr la aplicación de captura y detección.

Al pulsar el botón de configuración, accederá a un nuevo menú donde podrá rellenar las opciones de configuración.

También puede acceder a la ayuda general de la aplicación mediante el botón de ayuda.

Figura 66: Menú Captura

9.4.3. Iniciar Aplicación

La aplicación accede a la cámara del dispositivo y comienza con la captura de imágenes. Realiza una comparación entre imágenes, mediante un algoritmo de detección de movimiento basado en la distancia de Manhattan. En caso de producirse "movimiento" la imagen se envía SIEMPRE mediante un mensaje multimedia al terminal indicado en la configuración, y opcionalmente (ver Menú Inicio) se enviara por Internet.

Cuando se detecta movimiento la aplicación genera un periodo de pausa para evitar la saturación de imágenes similares, durante el cual no se enviará ninguna imagen, superado dicho espacio temporal, se seguirá ejecutando el algoritmo de captura y detección.

En caso de acceso accidental, por error, a la captura y detección, existe un comando "atrás" que parara la ejecución y volverá al menú anterior.

Para evitar el envío masivo de mensajes multimedia al iniciar la aplicación existe un cierto periodo de inicio de aplicación (ver Ubicación y tiempo de espera), durante el cual no se producirá ningún envío, aun estando iniciada la aplicación.

9.4.4. Menú Configuración

Dentro de este menú podrá configurar todas las variables necesarias para el correcto funcionamiento de la aplicación. La configuración de "palabras", puede realizarse de manera remota con el terminal receptor mediante el envío de mensajes de texto, si desea ahorrarse el coste de estos mensajes, basta con incluir en ambas configuraciones (captura y receptor) las mismas palabras de manera idéntica (formato, mayúsculas, puntuación, ortografía…). Los números de envíos serán propios del terminal y será IMPRESCINDIBLES para que se pueda enviar la imagen.

La primera vez que acceda a la aplicación deberá configurarla, el resto de accesos dicha configuración estará guardada en la memoria del teléfono, pudiendo modificarse cuando así lo desee.

Figura 67: Menú Configuración Captura

- Palabra inicio:

Dentro de esta pestaña dispone de un cuadro de texto donde introducir la palabra de inicio. Esta palabra se utilizará para el inicio remoto de la aplicación. Se guarda en la memoria interna del teléfono y podrá ser modificada en cualquier momento.

- Palabra fin

Dentro de esta pestaña dispone de un cuadro de texto donde introducir la palabra de fin. Esta palabra se utilizará para finalizar de forma remota la aplicación (ya sea vía SMS o vía bluetooth). Se guarda en la memoria interna del teléfono y podrá ser modificada en cualquier momento.

- **Palabra pausa**

Dentro de esta pestaña dispone de un cuadro de texto donde introducir la palabra de pausa. Esta palabra se utilizará para pausar de forma remota la aplicación. Se guarda en la memoria interna del teléfono y podrá ser modificada en cualquier momento.

Estas tres palabras pueden ser la misma (por razones de seguridad se recomienda que sean diferentes) pero no es una obligación impuesta por la aplicación.

Existe un caso de palabra invalida, cuando se introduce una palabra vacía (en blanco) se muestra un mensaje de error.

Mediante el botón confirmar datos se guarda en memoria la palabra. En caso de que no aparezca un mensaje de confirmación, repita la acción, puesto que se habrá producido un error en el almacenado de la palabra [ver Figura 68].

Figura 68: Menú Configuración Captura, palabra guardada

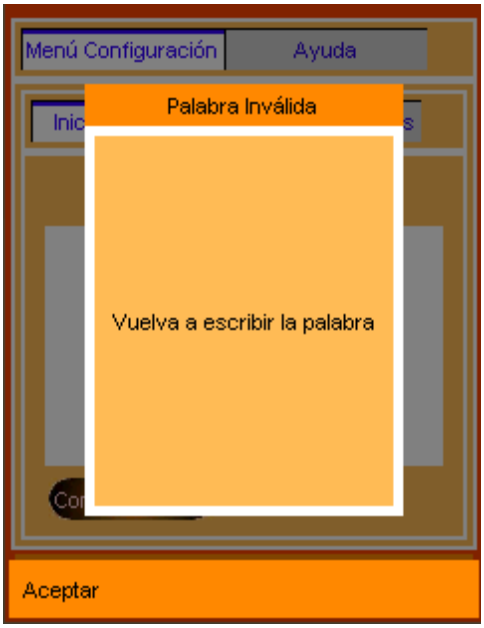

Figura 69: Menú Configuración Captura, palabra inválida

- Números envío

En esta pestaña aparecen tres cuadros de texto para introducir los tres posibles destinos[6] del mensaje multimedia.

Aparecerá un comando para guardar los números.

Los números han de ser diferentes entre sí y de nueve cifras, en caso de no ser así aparecerá un mensaje de error indicándolo.

A su vez el número principal siempre debe existir antes que los siguientes, y el número terciario solo se guardará en el caso de que existan los dos anteriores.

En caso de que no exista el número principal, se indicará mediante un mensaje de error.

[6] *Los tres posibles destinos no son obligatorios, solo lo es el numero principal, este dato es de suma importancia ya que cada numero supone el envío de un mensaje multimedia a ese destino, con los costes que ello conlleva.*

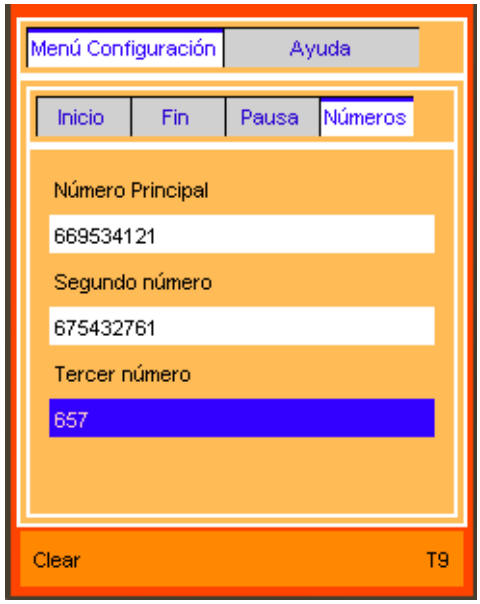

Figura 70: Menú Configuración Captura, números envío

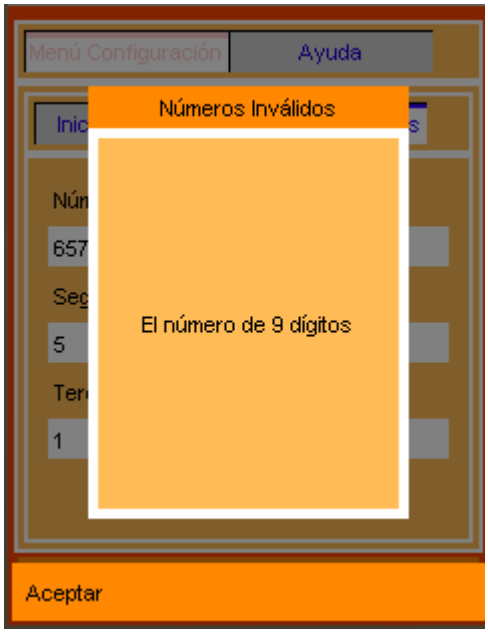

Figura 71: Menú Configuración Captura, números envío error (no 9 dígitos)

Figura 72: Menú Configuración Captura, números envío, error (números iguales)

Figura 73: Menú Configuración Captura, números envío, error (al menos número principal)

Dispone de un comando atrás para volver al menú anterior

9.4.5. Ubicación y tiempo de espera

Deberá seleccionar una situación adecuada del dispositivo teniendo en cuenta una serie de premisas:

- La orientación de la cámara: deberá apuntar a focos en los que el movimiento se produzca por una situación o agente externo no habitual, es decir, en puntos clave de seguridad. Preferentemente también en los que no se produzcan bruscos cambios de luz.

- Debe tener en cuenta que el dispositivo para un funcionamiento prolongado debe estar conectado a la red eléctrica, ya que la vida de la batería es bastante limitada.

Para poder ubicar el teléfono en el lugar adecuado sin que la aplicación comience la detección, existe un retardo de 30 segundos para que se inicie la aplicación, y así evitar el envío masivo de mensajes durante la colocación.

9.5. Receptor

9.5.1. Menú Receptor

Dentro de este menú podrá acceder a la configuración del dispositivo receptor y tendrá las opciones de control remoto del dispositivo de captura.

Figura 74: Menú Receptor

9.5.2. Menú Configuración

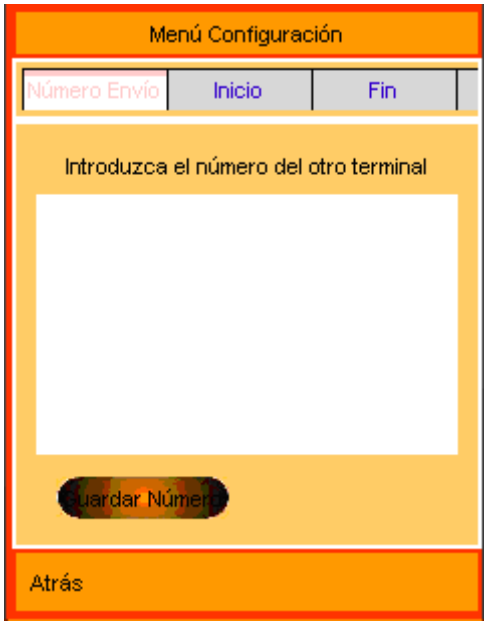

Figura 75: Menú Configuración Receptor (1)

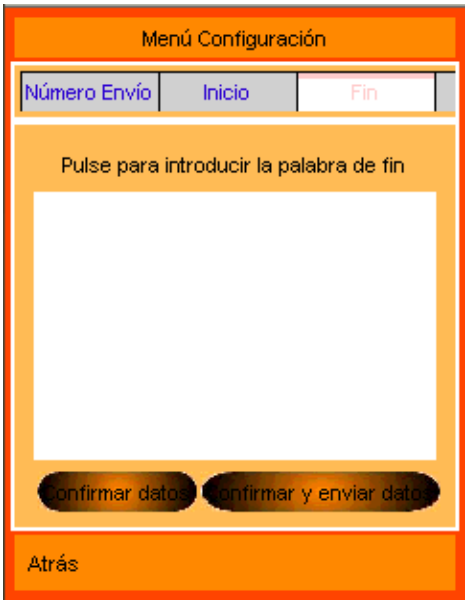

Figura 76: Menú Configuración Receptor (2)

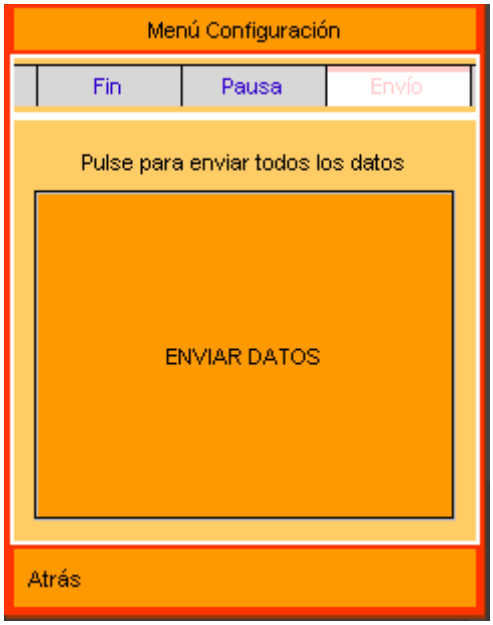

Figura 77: Menú Configuración Receptor (3)

- **Número envío:** Número de teléfono del dispositivo que va a utilizar el modo captura, se utilizará para el envío de las órdenes de manera remota. Mediante el botón "guardar número", se incluirá en la memoria del teléfono, siempre y cuando tenga el formato valido. En caso de ser erróneo aparecerá un mensaje de error. Si ya se ha introducido anteriormente al entrar en esta pantalla el número aparecerá en el cuadro de texto.

- **Palabra Inicio:** Palabra clave que se utilizará para iniciar la aplicación de manera remota.

- **Palabra Pausa:** Palabra clave que se utilizará para pausar la aplicación de manera remota

- **Palabra Fin:** Palabra clave que se utilizará para finalizar la aplicación de manera remota.

Estas tres palabras pueden ser la misma (por razones de seguridad se recomienda que sean diferentes) pero no es una obligación impuesta por la aplicación.

Existe un caso de palabra inválida, cuando se introduce una palabra vacía (en blanco) se muestra un mensaje de error.

Mediante el botón confirmar datos se guarda en memoria la palabra. En caso de que no aparezca un mensaje de confirmación, repita la acción, puesto que se habrá producido un error en el almacenado de la palabra (ver Figura 12).

Tras la confirmación se ofrece la opción de envío inmediato e individualizado de la palabra mediante un dialogo, si desea enviarlo pulse “SI”, en caso contrario, pulse “No”

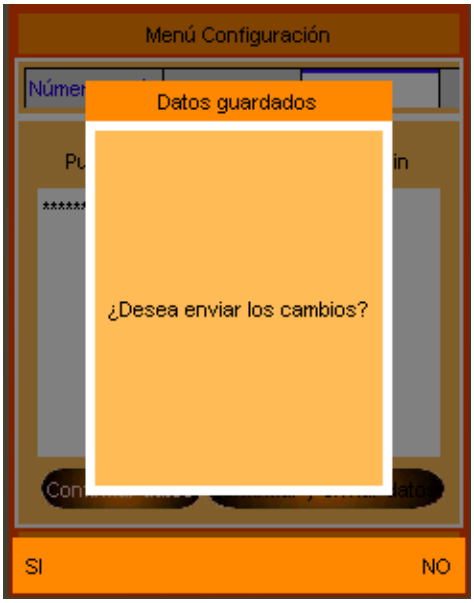

Figura 78: Menú Configuración Receptor, pregunta envío datos

También puede realizar el guardado y envío de la palabra de manera individual con un solo paso, mediante el botón "Confirmar y enviar".

Las palabras se enviarán con un mensaje de texto cifrado al número indicado en el apartado de "Numero de envío"

- **Envío Datos:** Si desea enviar las tres palabras con un solo mensaje de texto, en vez de enviar uno por palabra, pulse el botón "ENVIAR DATOS", situado en esta pestaña [ver Figura 76]

Mediante el comando atrás, volverá al menú anterior

9.5.3. Iniciar Aplicación

En caso de que exista el número de envío y la palabra de inicio, con este botón se enviará un mensaje de texto que al ser recibido por la aplicación de captura, iniciará la toma de imágenes y el análisis de movimiento.

En caso de no existir el número de envío aparecerá un diálogo indicándolo.

En caso de no existir la palabra de inicio aparecerá un diálogo indicándolo.

9.5.4. Pausar Aplicación

En caso de que exista el número de envío y la palabra de pausa, con este botón se enviará un mensaje de texto que al ser recibido por la aplicación de captura, se pausará, en caso de que este iniciada (con la posibilidad de reiniciarla de manera remota).

En caso de no existir el numero de envío aparecerá un dialogo indicándolo

En caso de no existir la palabra de pausa aparecerá un dialogo indicándolo.

9.5.5. Terminar Aplicación

Al seleccionar esta opción, aparecerá un cuadro de diálogo en el que podrá elegir la forma de terminar la aplicación de captura en el otro terminal.

- **Vía SMS:**

En caso de que exista el número de envío y la palabra de fin, con este botón se enviara un mensaje de texto que al ser recibido por la aplicación de captura, se cerrará, en caso de que este iniciada (sin posibilidad de reiniciarla de manera remota).

En caso de no existir el numero de envío aparecerá un dialogo indicándolo.

En caso de no existir la palabra de fin aparecerá un dialogo indicándolo.

- **Vía Bluetooth:**

En caso de que exista la palabra fin, mediante este sistema se cerrará la aplicación de captura, si ambos dispositivos se encuentran dentro del alcance. Sin ningún tipo de coste de envío.

Figura 79: Menú Configuración Receptor, Terminar aplicación

9.5.6. Uso

Esta aplicación se utiliza principalmente para la configuración y el control remoto del dispositivo de captura. Cuando se produce una detección la aplicación de captura enviará un mensaje multimedia al dispositivo receptor. El dispositivo receptor es capaz de tratar el mensaje multimedia estando o no arrancada la aplicación, si bien al estar arrancada la aplicación, la visualización de la imagen será directa.

9.6. Emulador

Para el uso dentro de un emulador en un PC deberá utilizar la aplicación situada en la carpeta “emulador”, sin necesidad de utilizar la aplicación “Características del dispositivo”.

Para el correcto funcionamiento en el emulador, arranque la aplicación de captura en el terminal +5550000, y la aplicación receptor en el terminal +5550001.

Se recomienda cambiar el vídeo que simula la captura por un en el que exista poco movimiento. El cambio ha de realizarse sustituyendo el vídeo de la siguiente ruta: C:\\”ruta donde esté instalado el emulador”\ WTK2.5.2\wtklib\media\viewFinder.

Recuerde que para el uso completo debe tener arrancados el servidor y la base de datos.

10. Bibliografía

[1] API Midp 2.0. http://java.sun.com/javame/reference/apis/jsr118/

[2]API MMAPI. http://java.sun.com/javame/reference/apis/jsr135/

[3] API RMS
http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/rms/package-summary.html

[4] API WMA. JSR 120 Expert Group. API (WMA) for Java™ 2 Micro Edition. Version 1.1.

[5] Borches Juzgado, Pedro Daniel. Java 2 Micro Edition: Soporte Bluetooth. PFC Carlos III de Madrid 20-Marzo-2004

[6] Bluecove. http://bluecove.org

[7] Campo Vázquez, Celeste y Tierno Alvite, Jonatan. Aplicaciones de tratamiento de imagen en terminales J2ME con cámara. Articulo Universidad Carlos III

[8] Control view, vigilancia en tiempo real.
http://www.controlview.es/proteccion.php (visitada 10-11-2009)

[9] Cortés, Ángel (2003). «30 años del primer móvil». www.Noticiasdot.com.

[10] Diccionario de la Real Academia de la Lengua Española. www.rae.es

[11] Digia. http://www.digia.com/C2256FEF0043E9C1/fwHome?readForm

[12] DigiaImageSpy.
http://www.allaboutsymbian.com/forum/showthread.php?t=5650

[13] Especificaciones técnicas HTC Magic. http://www.htc.com/es/product/magic/specification.html

[14] Especificaciones técnicas iPhone. http://www.apple.com/es/iphone/specs.html

[15] Especificaciones técnicas Nokia 3210. http://es.wikipedia.org/wiki/Nokia_3210

[16] Especificaciones técnicas Nokia N5800 Express Music. http://www.nokia.es/productos/moviles/nokia-5800-xpressmusic-pantalla-tactil-mp3-gps-camara-3g/especificaciones

[17] Especificaciones técnicas Nokia N97. http://www.nokia.es/productos/moviles/nokia-n97/especificaciones

[18] Especificaciones LWUIT. Developer's Guide. Lightweight UI Toolkit. Julio 2008 y Julio 2009

[19] Feng, Yu and Zhu, Jun. Wireless Java Programming with Java 2 Micro Edition.

[20] Froufe, Agustín y Jorge, Patricia. Ra-Ma J2ME : Java 2 micro edition : manual de usuario y tutorial.

[21] García, Carlos J2ME. Java Wireless Message API (WMA): manual.

[22] JavaWorld. http://www.javaworld.com/javaworld/jw-09-2007/jw-09-mobilevideo1.html?page=1. http://www.javaworld.com/javaworld/jw-09-2007/jw-09-mobilevideo2.html

[23] Lezama Lugo, A. Modelado de dispositivos para un sistema de seguridad implementado en tecnología JINI, Tesis licenciatura. 2001

[24] Mallick, Martyn. Mobile and Wireless Design Essentials

[25] MIDP APIs for Wireless Applications. A Brief Tour for Software Developers

[26] Sun Developers Network (SDN).
http://developers.sun.com/mobility/midp/articles/picture/

[27] Sun JavaTM Wireless Toolkit for CLDC, User's Guide. Version 2.5.2
September 2007

[28] SuperInventos.com, televigilancia y seguridad electrónica.
http://www.superinventos.com/televigilancia.htm

[29] SpyManager. http://www.8mobile.org/spyManager.aspx

[30] Vídeo Comando Actualidad, "Nos vigilan", RTVE
http://www.rtve.es/alacarta/player/626904.html. Noviembre 2009

[31] Vídeo LWUIT. http://www.youtube.com/watch?v=RzgWqOpNJIU

[32] Vídeo evolución terminales móviles
http://www.youtube.com/watch?v=sjt2NwNZZDM&feature=player_embedded

[33] Videovigilancia.com http://www.videovigilancia.com/televigilancia.htm
(visitada el 10-11-2009)

Aplicaciones:

[34] Características del dispositivo \ caracterisiticas-del_dispositivo\camara.jad

[35] Herramienta Device AnyWhere http://www.deviceanywhere.com/

[36] Ejemplo LWUIT \ejemplo\LWUITDemo\dist\LWUITDemo.jad